



UNIVERSIDAD CARLOS III DE MADRID

TESIS DOCTORAL

A Comprehensive Framework for the Rapid Prototyping of Ubiquitous Interaction

Autor:

D. Andrea Bellucci

Directores:

Dr. D. Ignacio Aedo Cuevas

Dr. D. Alessio Malizia

DEPARTAMENTO DE INFORMÁTICA
DOCTORADO EN CIENCIA Y TECNOLOGÍA INFORMÁTICA

Leganés, Octubre 2013

TESIS DOCTORAL

A COMPREHENSIVE FRAMEWORK FOR THE RAPID PROTOTYPING OF UBIQUITOUS INTERACTION

Autor: D. Andrea Bellucci

Directores: Dr. D. Ignacio Aedo Cuevas, Dr. D. Alessio Malizia

Firma del Tribunal Calificador:

Nombre y Apellidos

Firma

Presidente: Prof. D. Antonio de Amescua Seco

Vocal: Prof. D. Carmelo Antonio Ardito

Secretario: Prof. D. Juan Manuel Dodero Beardo

Calificación:

Leganés, de de 2013.

A mamma e papà.

Acknowledgments

IT is hard to express with words what does it feel like to have reached the end of this other phase in my life: getting a doctoral degree. It is often said that a picture is worth of a thousand words. The figure in the next page condensates my experience of pursuing a Ph.D. : the initial disorientation, the ordeal of getting stuck, the resolution to carry on no matter what, having faith that someday I will be able to see the bigger picture and, at the end, the feeling that I have done a good job or, at least, the very best that I could do. I will then, let a picture talk for me, hoping that the reader, the one who has already passed for the same path and the one who does not, can connect with my feelings. I know that anyone who has dedicated an important part of his life-time in a project will understand. Let me just quote one of my favorite authors, whose writing have deeply influenced my perspective about life. Nietzsche once said that *"what that does not kill us makes us stronger"*. That is what this dissertation means to me. These last four years have been blissfully hard. I please all the challenges that I have had to face: leaving the eternal city, settling in a new country, learning a new language, acquiring a new family, learning what does *doing research* means and how difficult is to find motivation within myself more than seek for it outside. I please all these challenges that have been instigating my intelligence all the time to find new ways to get rid of the obstacles. That is what life is for, standing up against adversity and learn from them as well as from pleasant moments. I feel stronger, now.

I would just like to add a few line to thank all the people that helped, directly or indirectly, to have made this dissertation possible. First, I would like to thank the supervisor of this thesis, Prof. Ignacio Aedo and the co-supervisor Prof. Alessio Malizia. Both of them made it possible that I could move from Italy to Spain and start this journey. From Nacho I have learnt the subtle art of diplomacy and how there are different ways to express the same concept. From Alessio, I have learnt that *"it's important in life not necessarily to be strong, but to feel strong"*. I would also like to thank Prof. Paloma Díaz, head of the DEI research group at Universidad Carlos III de Madrid. I have found in the DEI laboratory a place to nurture my research skills, discriminating what is the difference between *applied research* and *system development* in Computer Science. With the DEI research group I share the vision of digital technologies as a way to augment human capabilities and interaction as a way of framing the relationship between people and the objects designed for them.

The financial support from the Integra¹ and TIPEX² projects has been generous and has allowed me to both enjoy life in Madrid, without any economic concerns, and do research at the same time.

During my time as a Ph.D. student I have visited the Cardiff School of Art and Design and the National Center for Product Design and Development Research (PDR) at the Cardiff Metropolitan University in Cardiff, Wales, UK. I would like to thank Prof. Steve Gill for have given me the opportunity of working in a motivating and inspirational environment. Three months I have spent at Cardiff Metropolitan University and every day, before start working at the thesis, I took some moments to read the quote on the wall of my office at PDR — *"The timid and the fainthearted, and the people that expect quick results, are doomed to disappointment"* — until the point I converted it into a personal mantra. At Cardiff Metropolitan University I have had the possibility to experiment how the synergy of different disciplines, engineering, computer science and product design, can really benefit research. I would also like to thank Gareth Loudon to show me the bigger picture: the Ph.D. is an extensive training on learning how to cope with problems with my own forces and everything that I am learning now, I will use it in my life, someday. A special thank goes to Prof. Alan Dix, who invited me to the Tiree island, in the north of Scotland, to test some of the technologies I have been developing during the thesis. I would have never had imagined that my research pushed me so far...literally.

I would like to thank all my family for their support at a distance and for being always there anytime I needed to get back to my safe haven to recharge batteries. These years far from home have been the best occasion to put in practice everything mum and dad tried to transmit me.

I thank Juanma and Dani with whom I have shared the office and the joys and sorrows of chasing a doctoral degree. We have had good times sympathizing with each other about our dissertation. We also share the same taste for good movies...

One last, sincere and deep thank goes to my closest friends: Vanessa, Pedro, Marina, Carlos, Nuria, David y Luz. They are family to me. Through them I have learnt that transcendancy is in everyday, habitual, little actions.

¹CDTI, Spanish Ministry of Science and Innovation

²Spanish Ministry of Science and Innovation, TIN2010-19859-C03-01



Abstract

IN the interaction between humans and computational systems, many advances have been made in terms of hardware (e.g., smart devices with embedded sensors and multi-touch surfaces) and software (e.g., algorithms for the detection and tracking of touches, gestures and full body movements). Now that we have the computational power and devices to manage interactions between the *physical* and the *digital* world, the question is—*what should we do?* For the Human-Computer Interaction research community answering to this question means to materialize Mark Weiser's vision of Ubiquitous Computing.

In the desktop computing paradigm, the desktop metaphor is implemented by a graphical user interface operated via mouse and keyboard. Users are accustomed to employing artificial control devices whose operation has to be learned and they interact in an environment that inhibits their faculties. For example the mouse is a device that allows movements in a two dimensional space, thus limiting the twenty three degrees of freedom of the human's hand. The Ubiquitous Computing is an evolution in the history of computation: it aims at making the interface disappear and integrating the information processing into everyday objects with computational capabilities. In this way humans would no more be forced to adapt to machines but, instead, the technology will harmonize with the surrounding environment. Conversely from the desktop case, ubiquitous systems make use of heterogeneous Input/Output devices (e.g., motion sensors, cameras and touch surfaces among others) and interaction techniques such as touchless, multi-touch, and tangible. By reducing the physical constraints in interaction, ubiquitous technologies can enable interfaces that endow more expressive power (e.g., free-hand gestures) and, therefore, such technologies are expected to provide users with better tools to think, create and communicate.

It appears clear that approaches based on classical user interfaces from the desktop computing world do not fit with ubiquitous needs, for they were thought for a single user who is interacting with a single computing systems, seated at his workstation and looking at a vertical screen. To overcome the inadequacy of the existing paradigm, new models started to be developed that enable users to employ their skills effortlessly and lower the cognitive burden of interaction with computational machines. Ubiquitous interfaces are pervasive and thus invisible to its users, or they become invisible with successive interactions in which the users feel they are instantly and continuously successful.

All the benefits advocated by ubiquitous interaction, like the invisible interface and a more *natural* interaction, come at a price: the design and development of interactive systems raise new conceptual and practical challenges. Ubiquitous systems communicate with

the real world by means of sensors, emitters and actuators. Sensors convert real world inputs into digital data, while emitters and actuators are mostly used to provide digital or physical feedback (e.g., a speaker emitting sounds). Employing such variety of hardware devices in a real application can be difficult because their use requires knowledge of underneath physics and many hours of programming work. Furthermore, data integration can be cumbersome, for any device vendor uses different programming interfaces and communication protocols. All these factors make the rapid prototyping of ubiquitous systems a challenging task.

Prototyping is a pivoting activity to foster innovation and creativity through the exploration of a design space. Nevertheless, while there are many prototyping tools and guidelines for traditional user interfaces, very few solutions have been developed for a holistic prototyping of ubiquitous systems. The tremendous amount of different input devices, interaction techniques and physical environments envisioned by researchers produces a severe challenge from the point of view of general and comprehensive development tools. All of this makes it difficult to work in a design and development space where practitioners need to be familiar with different related subjects, involving software and hardware. Moreover, the technological context is further complicated by the fact that many of the ubiquitous technologies have recently grown from an embryonic stage and are still in a process of maturation; thus they lack of stability, reliability and homogeneity. For these reasons, it is compelling to develop tools support to the programming of ubiquitous interaction. In this thesis work this particular topic is addressed.

The goal is to develop a general conceptual and software framework that makes use of hardware abstraction to lighten the prototyping process in the design of ubiquitous systems. The thesis is that, by abstracting from low-level details, it is possible to provide unified, coherent and consistent access to interacting devices independently of their implementation or communication protocols. In this dissertation the existing literature is revised and is pointed out that there is a need in the art of frameworks that provide such a comprehensive and integrate support. Moreover, the objectives and the methodology to fulfill them, together with the major contributions of this work are described. Finally, the design of the proposed framework, its development in the form of a set of software libraries, its evaluation with real users and a use case are presented. Through the evaluation and the use case it has been demonstrated that by encompassing heterogeneous devices into a unique design it is possible to reduce user efforts to develop interaction in ubiquitous environments

Resumen

EN la interacción entre personas y sistemas de computación se han realizado muchos adelantos por lo que concierne el hardware (p.ej., dispositivos inteligentes con sensores integrados y superficies táctiles) y el software (p.ej., algoritmos para el reconocimiento y rastreo de puntos de contactos, gestos de manos y movimientos corporales). Ahora que se dispone del poder computacional y de los dispositivos para proporcionar una interacción entre el mundo *físico* y el mundo *digital*, la pregunta es—*que se debería hacer?* Contestar a esta pregunta, para la comunidad de investigación en la Interacción Persona-Ordenador, significa hacer realidad la visión de Mark Weiser sobre la Computación Ubicua.

En el paradigma de computación de escritorio, la metáfora del escritorio se implementa a través de la interfaz gráfica de usuario con la que se interactúa a través de teclado y ratón. En este paradigma, los usuarios se adaptan a utilizar dispositivos artificiales, cuyas operaciones deben ser aprendidas, y a interactuar en un entorno que inhibe sus capacidades. Por ejemplo, el ratón es un dispositivo que permite movimientos en dos dimensiones, por tanto limita los veintitrés grados de libertad de una mano. La Computación Ubicua se considera como una evolución en la historia de la computación: su objetivo es hacer que la interfaz desaparezca e integrar el procesamiento de la información en los objetos cotidianos, provistos de capacidad de cómputo. De esta forma, el usuario no se vería forzado a adaptarse a la máquina sino que la tecnología se integraría directamente con el entorno. A diferencia de los sistemas de sobremesa, los sistemas ubicuos utilizan dispositivos de entrada/salida heterogéneos (p.ej., sensores de movimiento, cámaras y superficies táctiles entre otros) y técnicas de interacción como la interacción sin tocar, multitáctil o tangible. Reduciendo las limitaciones físicas en la interacción, las tecnologías ubicuas permiten la creación de interfaces con un mayor poder de expresión (p.ej., gestos con las manos) y, por lo tanto, se espera que proporcionen a los usuarios mejores herramientas para pensar, crear y comunicar.

Parece claro que las soluciones basadas en las interfaces clásicas no satisfacen las necesidades de la interacción ubicua, porque están pensadas por un único usuario que interactúa con un único sistema de computación, sentado a su mesa de trabajo y mirando una pantalla vertical. Para superar las deficiencias del paradigma de escritorio, se empezaron a desarrollar nuevos modelos de interacción que permitiesen a los usuarios emplear sin esfuerzo sus capacidades innatas y adquiridas y reducir la carga cognitiva de las interfaces clásicas. Las interfaces ubicuas son *pervasivas* y, por lo tanto, invisibles a sus usuarios, o devienen invisibles a través de interacciones sucesivas en las que los usuarios siempre se sienten que están teniendo éxito. Todos los beneficios propugnados por la interacción ubicua, como la interfaz invisible o una interacción más *natural*, tienen un coste: el diseño

y el desarrollo de sistemas de interacción ubicua introducen nuevos retos conceptuales y prácticos. Los sistemas ubicuos comunican con el mundo real a través de sensores y emisores. Los sensores convierten las entradas del mundo real en datos digitales, mientras que los emisores se utilizan principalmente para proporcionar una retroalimentación digital o física (p.ej., unos altavoces que emiten un sonido). Emplear una gran variedad de dispositivos hardware en una aplicación real puede ser difícil, porque su uso requiere conocimiento de física y muchas horas de programación. Además, la integración de los datos puede ser complicada, porque cada proveedor de dispositivos utiliza diferentes interfaces de programación y protocolos de comunicación. Todos estos factores hacen que el prototipado rápido de sistemas ubicuos sea una tarea que constituye un difícil reto en la actualidad.

El prototipado es una actividad central para promover la innovación y la creatividad a través de la exploración de un espacio de diseño. Sin embargo, a pesar de que existan muchas herramientas y líneas guías para el prototipado de las interfaces de escritorio, a día de hoy han sido desarrolladas muy pocas soluciones para un prototipado holístico de la interacción ubicua. La enorme cantidad de dispositivos de entrada, técnicas de interacción y entornos físicos concebidos por los investigadores supone un gran desafío desde el punto de vista de un entorno general e integral. Todo esto hace que sea difícil trabajar en un espacio de diseño y desarrollo en el que los profesionales necesitan tener conocimiento de diferentes materias relacionadas con temas de software y hardware. Además, el contexto tecnológico se complica por el hecho que muchas de estas tecnologías ubicuas acaban de salir de un estadio embrionario y están todavía en un proceso de desarrollo; por lo tanto faltan de estabilidad, fiabilidad y homogeneidad. Por estos motivos es fundamental desarrollar herramientas que soporten el proceso de prototipado de la interacción ubicua. Este trabajo de tesis doctoral se dedica a este problema.

El objetivo es desarrollar una arquitectura conceptual y software que utilice un nivel de abstracción del hardware para hacer mas fácil el proceso de prototipado de sistemas de interacción ubicua. La tesis es que, abstrayendo de los detalles de bajo nivel, es posible proporcionar un acceso unificado, consistente y coherente a los dispositivos de interacción independientemente de su implementación y de los protocolos de comunicación. En esta tesis doctoral se revisa la literatura existente y se pone de manifiesto la necesidad de herramientas y marcos que proporcionen dicho soporte global e integrado. Además, se describen los objetivos propuestos, la metodología para alcanzarlos y las contribuciones principales de este trabajo. Finalmente, se presentan el diseño del marco conceptual, así como su desarrollo en forma de un conjunto de librerías software, su evaluación con usuarios reales y un caso de uso. A través de la evaluación y del caso de uso se ha demostrado que considerando dispositivos heterogéneos en un único diseño es posible reducir los esfuerzos de los usuarios para desarrollar la interacción en entornos ubicuos.

Contents

Acknowledgments	iii
Abstract	vi
Resumen	viii
1 Introduction	1
1.1 Interaction	4
1.2 Scope	6
1.3 Motivation	8
1.4 Methodology	10
1.4.1 Problem	11
1.4.2 Objectives	12
1.4.3 Designing and Developing the Artifact	13
1.4.4 Testing and Evaluating	14
1.4.5 Communicating the Results	15
1.5 Outline	16
2 State of the Art	17
2.1 Device Ecologies	18
2.1.1 Physical Computing	20
2.2 Touchless/Remote Interaction Technologies	20
2.2.1 Implications for Interaction Design	26
2.3 Multi-touch Interaction Technologies	27
2.3.1 Implications for Interaction Design	31
2.4 Tangible Interactive Technologies	32
2.4.1 Implications for Interaction Design	35

2.5	Libraries, Toolkits and Frameworks for Ubiquitous Interaction	35
2.5.1	Library, Toolkit and Framework	36
2.5.2	Touchless/Remote Interaction	37
2.5.3	Multi-Touch Interaction	40
2.5.4	Tangible Interaction	44
2.5.5	Physical Computing	47
2.6	Rapid Prototyping	51
2.6.1	The Role of Prototyping in HCI	52
2.6.2	Rapid Prototyping for Ubiquitous Interaction	53
2.7	Summary	59
3	Open Issues	61
4	Exploration of the design space	64
4.1	Experiences from the Development of Real Systems	64
4.1.1	Remote Interactions for Screen Displays with the Wiimote	65
4.1.2	Multi-touch Interactions around a Table with the DiamondTouch	68
4.1.3	TESIS: Turn Every Surface Into an Interactive Surface	70
4.2	Requirements for a Framework Supporting Ubiquitous Interaction	74
4.2.1	Requirements Elicitation: Interviews	76
4.2.2	Defining Categories from Requirements	81
4.2.3	Requirements	83
4.3	Summary	91
5	Design and Implentation of the Framework	93
5.1	Interaction Model	94
5.2	Architecture	103
5.2.1	Hardware Abstraction	105
5.2.2	Input Transformation	111
5.2.3	Input Interpretation	112
5.2.4	Application	113
5.2.5	Event propagation	114

5.2.6	Implementation	114
5.3	Compliance with requirements	116
5.4	Summary	118
6	Evaluation	120
6.1	Evaluation Background	120
6.1.1	The five themes	122
6.2	Case study: digitally-augmented product shelf	124
6.2.1	Goals	126
6.2.2	Discussion	127
6.3	User test	128
6.3.1	Goals	128
6.3.2	Participants	129
6.3.3	Design	129
6.3.4	Results and Discussion	136
6.4	Summary	151
7	Conclusions	152
7.1	Contributions	154
7.2	Potential for Future Research	155
	References	157
A	Published Results	169
B	Requirements Extraction	170
B.1	List of Initial Requirements	170
B.2	Interviews	172
B.3	Additional Requirements	173
B.4	Categories	173
B.5	Final Requirements Under Categories	174

C	User Evaluation	177
C.1	Documentation for the User Tasks	177
C.2	Written Consent Form	182
C.3	Pre-test questionnaire	184
C.4	Post-test questionnaire	188

List of Figures

1.1	The Nintendo Wii Remote Controller as part of the user interface.	2
1.2	A typical UbiComp environment for meeting rooms.	5
1.3	The context of the this research work.	7
1.4	The six-steps design and development research approach.	11
1.5	The modified six-steps design and development research approach.	11
2.1	A sketch of CodeSpace meeting environment. [Bragdon et al., 2011] . . .	19
2.2	Put-That-There [Bolt, 1980].	21
2.3	Minority Report: envisioning a touch-free interface.	22
2.4	Nintendo Wii Remote Controller.	23
2.5	SixthSense technology developed at Fluid Interfaces Group, MIT Media Lab [Mistry and Maes, 2009].	24
2.6	G-Speak interactive environment from Oblong Industries.	25
2.7	FTIR configuration for multi-touch displays.	28
2.8	Augmented Surfaces [Rekimoto and Saitoh, 1999].	29
2.9	DiamondTouch interactive tabletop.	31
2.10	Bricks. Source [Fitzmaurice et al., 1995].	33
2.11	metaDESK. Source [Ullmer and Ishii, 1997].	33
2.12	Designers' Outpost.	33
2.13	Tangible interface of the AudioPad system.	34
2.14	reactTable* tangible objects and topological interface.	34
2.15	OpenNI software architecture.	39
2.16	MT4j reference architecture.	43
2.17	The Papier Mache monitoring interface. Source [Klemmer et al., 2004]. .	45
2.18	reactIVision framework diagram.	46
2.19	The libTISCH architecture.	47

2.20	Phidgets' physical components.	49
2.21	d.tools authoring environment (left) and hardware components (right). . .	49
2.22	Arduino Uno board.	50
2.23	Squidy visual programming workspace.	56
2.24	Core classes of the ROSS API.	57
4.1	Don't Touch Me User Interface.	67
4.2	Visualization of the Widget for the civil protection role.	70
4.3	The TESIS system: a pico-projector, a depth-sensing camera and a laptop. .	71
4.4	Visualization of the recognition algorithm. From left to right, top to down: grayscale depth image, actual image after subtracting the background, contours detection and blob detection.	72
4.5	A fixed version of the TESIS prototype at the rural center in Tiree, Scotland, UK.	73
4.6	The process to define requirements for the framework.	75
4.7	Captures of student and lecture views of the ALF system. Source [Zarrao- nandia et al., 2012]	84
4.8	MT4j reference architecture.	88
4.9	TUI-VR reference architecture.	88
5.1	Main components of the framework in a real scenario.	94
5.2	MCRpd interaction model for Tangible Interaction. Source [Ullmer and Ishii, 2000].	95
5.3	Physical icons in metaDESK are coupled with digital information (e.g., the map of the MIT campus).	96
5.4	Physical objects and digital representations in the reacTable*.	97
5.5	The framework interaction model for device ecologies. Core classes (<i>Environment</i> , <i>PObject</i> , <i>TObject</i> and <i>DObject</i>) and their interconnections are depicted.	98
5.6	UML Class diagram for the proposed interaction model: <i>TObject</i>	99
5.7	Schematic setup of Example 1: the motion sensing capabilities of an iPad (accelerometer) are used to change the inclination of a physical board (implemented with a Lego Mindstorms NXT 2.0 set) and a digital horizon widget displayed on the screen of a laptop device.	100

5.8	Schematic setup of Example 1.	101
5.9	Scenario for Example 2: touching the display of the iPad on the virtual button will make the digital 3D box to change its color.	102
5.10	Schematic setup of Example 2.	102
5.11	The architecture of the framework.	104
5.12	The structure of the XML file corresponding to a <i>TObject</i>	106
5.13	Client/Server architecture of <i>TOBjects</i>	109
5.14	Input Transformation mechanisms in a <i>TObject</i>	112
5.15	Part of the source code for the scenario.	115
6.1	The digitally-augmented product shelf use case in the framework architecture.	125
6.2	Touch-enabled User Interface.	126
6.3	Excerpt of wrj4P5 API documentation.	134
6.4	An example of the template provided to the subjects.	135
6.5	Pre-test questionnaire, Q1: <i>"Of the following programming language and technologies, check those that you have personally used and are familiar with"</i>	136
6.6	Technical programming skills of the sample population. Q2, in dark green: <i>"Please rate your technical programming knowledge (programming paradigms, data structures, frameworks, etc.)"</i> . Q3 in light green: <i>"If you know/use the Java programming language, what is your level of proficiency with the language?"</i> . Box-plot on the left and distribution of the answers on the right.	137
6.7	Q8: <i>"Have you ever programmed interactive systems that make use of the aforementioned (Q7) hardware?"</i>	137
6.8	Q9: <i>"Is your academic/professional background related to interaction design, human-computer interaction or ubiquitous computing?"</i>	137
6.9	Box plot of the overall rating for the two libraries (on the left and red, Q2) and for the framework (on the right and blue, Q1).	138
6.10	Direct comparison of the overall rating for the two libraries (red, Q2) and the framework (blue, Q1).	139
6.11	Q4: <i>"How did the technical environment (Processing) make the programming tasks?"</i> Box plot on the left and histogram on the right.	140
6.12	Q5: <i>"Would you use the framework to develop interactive systems for ubiquitous environments?"</i> Box plot on the left and histogram on the right.	140

6.13	Threshold and Ceiling factor (Q6 to Q14).	141
6.14	Predictability factor (Q17 to Q21).	142
6.15	Moving Target factor (Q23 and Q24).	143
6.16	Comparison of Q24 and Q25 that shows how changing the input device (Wiimote and iPad) affected the development WITH (on the left and blue, Q24) and WITHOUT (on the right and red, Q25) the framework. 1 meaning "Drastically" and 7 meaning "Not at all".	144
6.17	Summary of the post-test questionnaire results using the scores for the three factors defined by Myers et al. [2000]: <i>Threshold and Ceiling</i> , <i>Predictability</i> and <i>Moving Target</i> . A fourth factor, <i>Overall</i> measures the overall acceptance of the framework.	145
6.18	Total time for each participants to accomplish the two tasks WITH (in blue) or WITHOUT (in red) the framework. On the x-axis the participant and on the y-axis the time are plotted.	146
6.19	Average times for each task and condition. Blue: Wiimote w/ framework. Red: iPad w/ framework. Green: Wiimote w/o framework. Purple: iPad w/o framework.	147
6.20	Frequency histogram of the two time series: WITH (in blue, on the left) and WITHOUT (in red) the framework.	148
6.21	Q-Q plot for the two time series: WITH (in blue) and WITHOUT (in red) the framework.	149

List of Tables

2.1	Examples of development kit and libraries for toucheless interaction. . . .	38
2.2	Examples of development kit and libraries for multi-touch interaction. . .	41
2.3	Examples of toolkits and frameworks for tangible interaction.	45
2.4	Examples of toolkits for physical computing.	48
2.5	Examples of frameworks and visual environments for ubiquitous interaction.	55
4.1	Main characteristic of the respondents population according to target users.	79
5.1	Attributes of the <i>tobject</i> tag.	105
5.2	Attributes of the <i>connection</i> tag.	107
5.3	Abstract data type hierarchy. The first column presents the abstract data type, the second column an example of a multi-dimensional extension of the data type and the third column an example of a device generating such input.	110
5.4	Spatial relationships between two entities <i>a</i> and <i>b</i>	113
5.5	Requirements compliance matrix. Green: fully addressed. Yellow: partially addressed. Red: future works.	118
6.1	Combinations of the two factors <i>Software Technology</i> and <i>Hardware Technology</i> . F_w : Framework with Wiimote, F_i : Framework with iPad, L_w : Library (wrj4P5) with Wiimote, L_i : Library (oscP5) with iPad.	130
6.2	The final runs of the experiments: combination of <i>Software and Hardware Technology</i> with <i>Interaction Component</i> . The letters represent each run of a 8x8 Latin Square. The development of the button was tested in Task 1 (T1) and the accelerometer in Task 2 (T2).	130
6.3	The 8x8 Latin Square for counterbalancing.	131

List of Abbreviations

AR	Augmented Reality, page 7
DIY	Do-It-Yourself, page 50
EUD	End-User Development, page 50
FTIR	Frustrated Total Internal Reflection, page 28
GUI	Graphic User Interface, page 8
HCI	Human-Computer Interaction, page 6
I/O	Input/Output, page 1
IDE	Integrated Development Environment, page 50
IR	InfraRed, page 23
IxD	Interaction Design, page 6
LED	Light Emitting Diode, page 63
MEMS	Micro-Electro-Mechanical Systems, page 3
NUI	Natural User Interface, page 22
PCB	Printed Circuit Board, page 48
PDA	Portable Digital Assistant, page 27
RFID	Radio Frequency IDentification, page 6
RGB	Red Green Blue colorspace, page 25
RGBD	RGB cameras plus Depth sensor, page 25
SDK	Software Development Kit, page 37
TIR	Total Internal Reflection, page 28
ToF	Time-of-Flight, page 25

TUI	Tangible User Interface, page 32
TUIO	Tangible User Interface Objects, page 12
UbiComp	Ubiquitous Computing, page 2
UDP	User Datagram Protocol, page 72
USB	Universal Serial Bus, page 48
VR	Virtual Reality, page 7
WIMP	Windows, Icons, Menus and Pointer, page 23

Conventions

Throughout this thesis proposal the following conventions are used:

- Unidentified third persons are always described in male form. This is only done for readability purposes.
- Links to websites of mentioned libraries, toolkits, frameworks, applications or interactive artifacts are shown in a footnote at the bottom of the corresponding page.
- The definition of Human-Computer Interaction (HCI) is a synonym for Human-Machine Interaction (HMI) as well as Man-Machine Interaction (MMI).
- References follow the ACM Computing Surveys citation format.
- All Web resources URLs have been lastly checked on date 27th of June, 2013.
- The whole thesis proposal is written in American English, with the abstract both in American English and Castilian Spanish (Castellano).

1

Introduction

Shall we begin like in David Copperfield?

I am born...

—*Interview with the vampire*

LOUIS, THE VAMPIRE.

WITH the third wave in the history of computing, the perception and meaning of user interaction is changing. In the same way desktop computers replaced bulky mainframes, starting from the last decade of the past century we assisted to the rise of Ubiquitous Computing [Weiser, 1991], which introduced new interaction techniques and Input/Output (I/O) technologies from multi-touch [Buxton, 2009, 2010; Buxton et al., 1985], to touchless or remote [de la Barré et al., 2009], to tangible [Fitzmaurice et al., 1995; Ishii, 2008], to multimodal [Oviatt, 2003]. With the advent of Ubiquitous Computing, digital information processing has come out from personal computers and it has become more and more integrated into everyday objects and activities. Therefore, in such environment pervaded by digital technologies, *"we became part of the interface or rather we bring the interface with us everywhere, we create practices around the interface"*, as Christensen [2006, p. 76] pointed out. The concept of interface between humans and computational machines embraces now physical devices or environmental elements augmented with digital capabilities. This makes embodied interactions possible—to exploit these new technologies, designers are currently trying to provide new ways for the users to interact with the surroundings in order to avoid the intrinsic limitations of the mouse and to promote a more *natural* approach [van Dam, 1997]. The Nintendo Wii Remote¹ controller (see Figure 1.1), the human fingers, the voice and even the whole human body are part of the user interface, not only mice and keyboards we use to interact with graphical elements visualized on a vertical display screen. This same vision has been previously shared by Jørgensen and Myers [2008] who stated that present interactive systems include *"tangible artifacts and conceptual entities embedded in the everyday world"*.

¹<http://wiibrew.org/wiki/Wiimote>

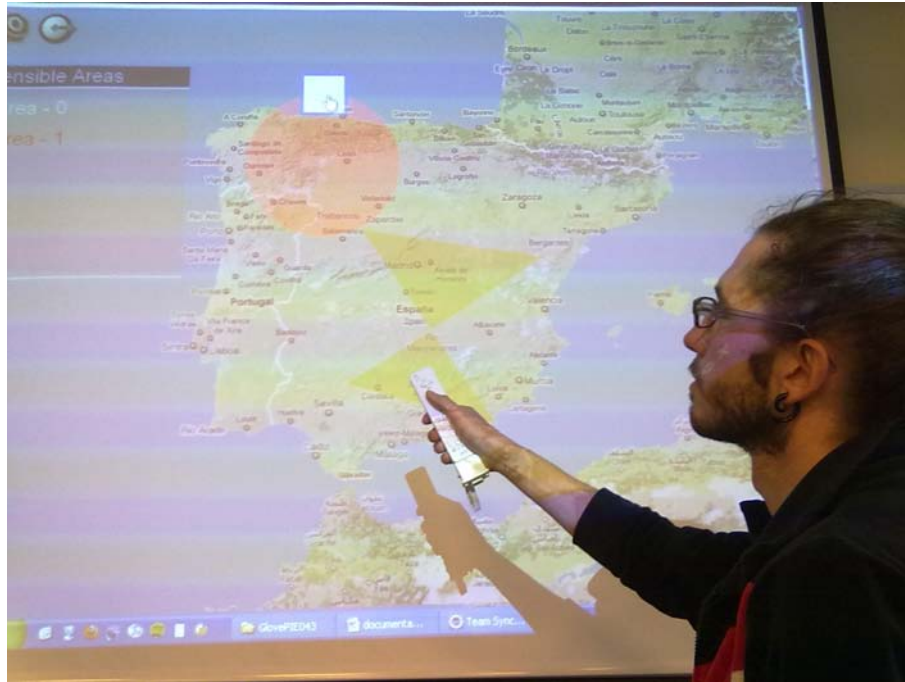


Figure 1.1: The Nintendo Wii Remote Controller as part of the user interface.

Ubiquitous Computing (also abbreviated UbiComp) is, therefore, a paradigm for Human-Computer Interaction that is considered as an advancement of the desktop computing model. A synthetic and neat definition of UbiComp has been given by York and Pendharkar [2004]:

[...] machines that fit the human environments instead of forcing humans to enter theirs.

The concept of computational devices merged into a real environment was firstly depicted by Weiser [1991] with an early suggestion of the world as interface to computing, referred to by him and his colleagues as *embedded virtuality*. In this context, UbiComp clearly stimulates the development of an interaction that is *beyond the desktop* and where digital bits are blended into objects of the physical world, as Weiser [1991] specifically pointed out in his seminal work:

[...] Ubiquitous Computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives.

Weiser envisioned the real world to be pervaded by a wide range of computational devices that harmonize with the surrounding environment in a way that it feels natural to the users

to interact with them, making therefore the interface become invisible. During normal activities a user, who is interacting with an ubiquitous system, may employ different I/O devices and artifacts at the same time and he may not necessarily be aware that he is doing so. The objective is to make the interaction with computers seamless and easy: less effort and cognitive burden reduces the sense of frustration and information overload typical of the interaction with desktop computers [Norman, 1999]. To achieve his vision, Weiser proposed a first taxonomy of UbiComp devices to be made of the three forms:

- **Tabs**, wearable devices with a centimeter-scale physical size. Belong to this category devices such as smartphones (e.g., the Apple iPhone²).
- **Pads**, hand-held devices with a decimeter-scale physical size. In this category fall examples of ultraportable or personal computers (e.g., the Apple's iPad³).
- **Boards**, interactive display devices with a meter-scale physical size. Examples are interactive tabletop surfaces such as the reacTable* [Jordà et al., 2007].

These three categories proposed all share some common aspects, such as being macro-sized, having a planar form and incorporating displays for visual output. During the last years, researchers have suggested to relax each one of these three attributes (the size, the form and the output display) in order to explore and define further possibilities for the computational devices to be actually merged into the real world. In particular, Poslad [2009] proposed three additional forms for ubiquitous systems, which are:

- **Dust**, miniaturized devices that can be without visual output displays, ranging from nanometres through micrometers to millimetres. These devices are commonly known as MicroElectroMechanical Systems (MEMS). Examples are digital and mechanical sensors that can detect variation in the surrounding environment and can be wirelessly networked and distributed over some area to perform tasks [Kahn et al., 1999].
- **Skin**, organic devices fabricated with light emitting and conductive polymers with computational capabilities. They can be exploited to provide flexible non-flat display surfaces. In this context MEMS devices can also be integrated into physical surfaces so that real world structures can act as networked surfaces of MEMS.
- **Clay**, MEMS combined into physical artifacts with connection to the digital world. Users can exploits such objects to interact with a computational system using real world affordances. Tangible User Interfaces are examples of this category.

²<http://www.apple.com/iphone/>

³<http://www.apple.com/ipad/>

The visionary contribution of Weiser [1991] opened the door to many inspirational research works towards an interaction between the human and the surrounding physical/virtual environment that would be less like the current desktop keyboard-mouse-display paradigm. Interaction in ubiquitous systems is inherently multimodal and it provides a range of techniques and tools to access, manage and share digital information. It is important to highlight that ubiquitous technologies have been thought to be human centered: they have to permeate the real world, helping people to achieve goals and fulfill their needs with minimum effort by exploiting different input modalities. Aiming at enabling people to interact in a less constrained way, new interaction techniques have been created that try to exploit the possibilities offered by the combination of *real world* affordances with their *digital* counterpart. These input modalities can be grouped in three main categories, namely: touchless/remote, multi-touch and tangible interaction. In the next chapter (Chapter 2), an overview of each category is given and a review of the literature is presented focusing on the outputs (from both the academia and the industry) that best express the potentials and limitations of each technology. By doing this, valuable insights are derived on the wide range of I/O devices that researchers and designers have to deal with in a UbiComp scenario.

1.1 Interaction

The Ubiquitous Computing is considered as an extension of the computational capabilities of the physical environment, allowing the computational infrastructure to be present everywhere in the form of small, inexpensive, robust networked processing devices, distributed at all scales throughout everyday life and generally turned to distinctly common-place ends. The design and placement of these devices have to be conceived according to users tasks and the context of interaction, as happens for context-aware applications where devices can both sense and react based on their environment [Dey et al., 2001]. Therefore, computation cannot be localized in a single point (e.g., the desktop terminal) but it is extended to different spots of the real world setting. Technologically-enhanced spaces are a manifestation of this concept of ubiquitous environments. They are physical spaces where the affordances of physical objects are augmented with digital capabilities thus creating an *ecology* of heterogeneous networked devices—*device ecology* is the keyword used in the literature to define such collection of different devices with relationships among each other [Jung et al., 2008]. A typical UbiComp environment for meeting rooms (see Figure 1.2), for example, would consist of a wall display screen operated via remote devices and touch or touchless hand gestures. Such gestures might be recognized by optical and electrical sensors placed in strategic points inside the room. Other users can be seated at a table and interact with a digital horizontal surface via touch gestures or using tangible input elements. At the same time, users could also take notes with their personal tablet devices, perhaps using a digital pen or through touch gestures



Figure 1.2: A typical UbiComp environment for meeting rooms.

with their fingers. These personal display devices can be connected with the shared displays so that if a user wants to share documents with other attendants, he can simply send the digital information from his device to the public display with the swipe of his fingers. A real implementation of an ubiquitous workspace is the i-LAND project [Streitz et al., 1999] from the Fraunhofer's Integrated Publication and Information Systems Institute (IPSI) in Darmstadt. In this project, different interactive devices coexist in the same physical space to support collaborative human work: a wall-sized display screen (the DynaWall), an interactive horizontal surface (the InteracTable) and two computer-augmented chairs with pen-based computer display and laptop docking. The scenario depicted by the previous example, like in the i-LAND project, establishes a socio-technical system [Mumford, 2000]. That is, it involves both human- and technological-related factors:

- From the point of view of **humans**, in the desktop computing paradigm, users interact via a mouse and a keyboard with a display screen placed on a desk: the interaction is limited and "*sensory deprived*"—using Negroponte's words⁴. In UbiComp systems, users are free to interact with devices using different techniques, such as touchless or remote, multi-touch or tangible interaction. These techniques provide a higher expressive power and enable users to employ their skills effortlessly and lower the cognitive burden of interaction with computational machines. Never-

⁴<http://www.wired.com/wired/archive/4.08/negroponte.html>

theless, seamless and graceful interaction is not only a matter of the interaction technique in use. Interaction is *natural* only when supported by an interface that *"makes learning enjoyable and eliminates the druggery that distracts from skilled practice"* [Wigdor and Wixon, 2011]. Moreover, people use digitally augmented devices in their everyday life to perform personal and collaborative activities and it is therefore important to understand how to best design such device ecologies in order to support users personal needs and social interactions.

- From the point of view of **machines**, heterogeneous devices coexists in the same environments. These devices ranges from (a) interactive surfaces such as digital whiteboards, large projected vertical surfaces and interactive tabletops, (b) personal devices such as smartphones or tablet with private display screens and, (c) I/O devices like physical objects enhanced with digital capabilities, touch sensors or motion tracking cameras. To make devices communicate among each other, interoperability is a paramount issue. For instance, the integration of dat from heterogeneous devices can be cumbersome in real applications, for any device vendor uses different interfaces and protocols. In order to exchange information they have to agree on common communication protocols and network architectures. Moreover, being operated with different interaction styles, they also need to share a common definition of the type of data they can process and transmit.

1.2 Scope

The particular research area addressed by this dissertation is Ubiquitous Interaction: interactions that take place in technologically-enhanced spaces as described in the previous Section. More formally, Ubiquitous Interaction has been defined as *"interaction in pervasive, ubiquitous, tangible, or ambient computing - including interaction with multiple, dynamic, and distributed interfaces"* [Klokmose, 2009]. Even if Ubiquitous Interaction is characterized by both human and technological factors, this work focuses on the technological aspects of interaction and it specifically copes with the issues of developing the infrastructure to support human collaborations through device ecologies. Figure 1.3 shows how this research embraces several aspects that pertain to the intersection of the three broader fields of Ubiquitous Computing (UbiComp), Human-Computer Interaction (HCI) and Interaction Design (IxD). From the perspective of UbiComp and HCI, the notion of Ubiquitous Interaction constitutes a new paradigm to develop interfaces that bridge the gap between the physical and the digital world and make it possible to enhance everyday objects with digital information processing [Ishii and Ullmer, 1997]. The development of new technologies (e.g., Radio Frequency IDentification or RFID, touch-enabled surfaces) and interaction techniques (e.g., multi-touch or touchless) has allowed to bring computing into the *'real world'* and outside the screen-based interaction of traditional

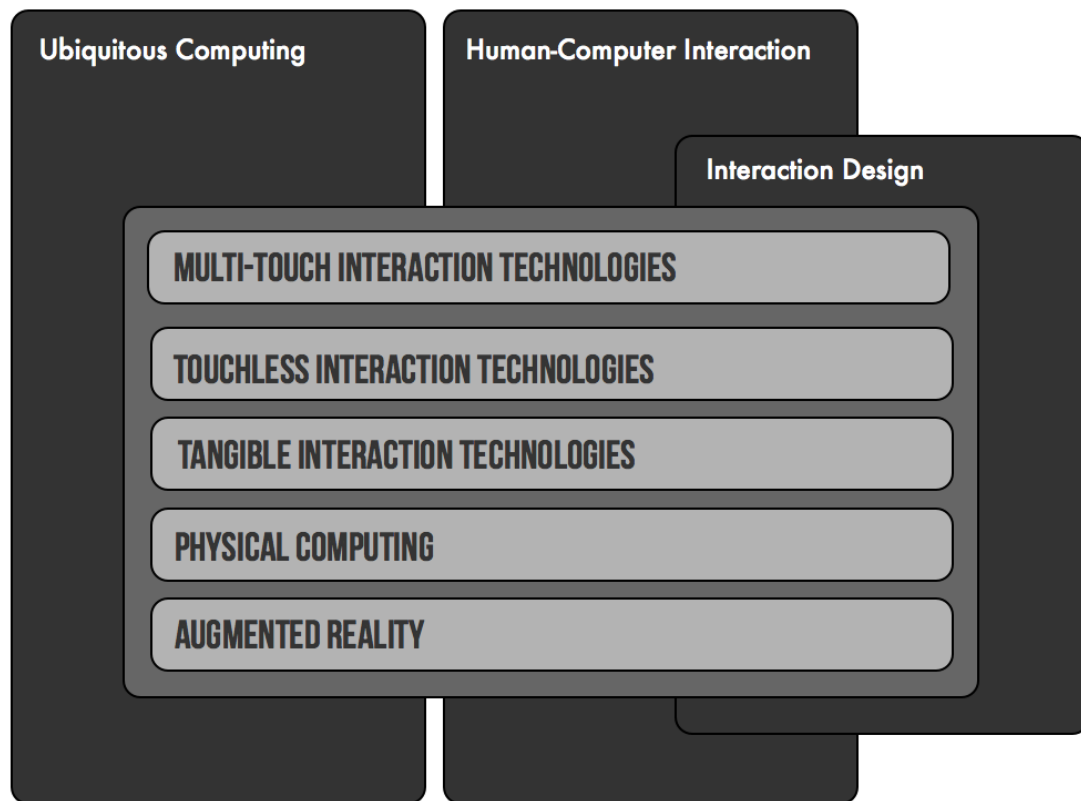


Figure 1.3: The context of the this research work.

desktop or Virtual Reality (VR) environments, which were considered as alienating technologies. In this sense, Ubiquitous Interaction is an umbrella term that comprises various research and design approaches to interaction such as Augmented Reality (AR) and Multi-touch, Touchless/Remote and Tangible Interaction Technologies. In Chapter 2 main characteristics and peculiarities of each interaction technology are discussed in details as well as common intersections and their implications in the development of interactive environments. Considering Interaction Design, the technological development presents new challenges and opportunities, being new interactive products more and more embedded with electronic circuits and sensors. The combination of physical and digital elements in design has been fostered by the widespread development of Physical Computing. According to Hornecker [2009], "[Physical Computing] involves fast prototyping with electronics, and often reuses and scavenges existing technology (tinkering). It is defined as the design of interactive objects, which are controlled by software, and that people interact with via sensors and actuators". Interaction Design also influenced and has been influenced by Tangible Input Technologies. For example, while at the beginning tangible objects have been mainly used as alternative input devices, IxD researchers have started to devise that digital I/O behaviors can be strongly coupled

with physical artifacts in such a way to provide rich interactions with digital systems. For instance, novel tangible interfaces allows the user manipulate the physical object in a way that changes in its physical characteristics produce a direct input to the digital system. As an example, *Beyond* [Lee and Ishii, 2010] is a collapsible tangible device, in the form of a stylus, for direct 3D manipulations. When pressed against the surface, the device collapse in the physical world and it extends into the digital, so that the user can have the illusion that they are inserting the tool into the virtual space.

1.3 Motivation

New interactive technologies and techniques are able to empower users with an higher degree of expressiveness. Computational devices are integrated in our living environment and, therefore, there is a need of an interaction that allows to seamlessly bridge between the physical and the digital world. One example is the vision of Tangible Bits as exposed by Ishii [2008], in which physical objects are integrated with interactive surfaces and coupled with digital information. Notwithstanding the rhetoric on the invisible interface and devices that harmonize with the environment, Oulasvirta [2008] describes how, in reality, UbiComp is facing a big divide: due to the heterogeneity of its technologies, practical applications and interfaces are pluggable and interchangeable only to a very limited extent. In particular, he reported that:

present-day IT infrastructure, “the real ubicomp,” is a massive noncentralized agglomeration of the devices, connectivity and electricity means, applications, services, and interfaces, as well as material objects such as cables and meeting rooms and support surfaces that have emerged almost anarchistically, without a recognized set of guiding principles. This infrastructure is not homogenous or seamless, but fragmented into several techniques that the user has to study and use.

Developers of ubiquitous interaction systems are experiencing the same difficulties that Graphic User Interface (GUI) designers encountered more than twenty years ago. At the beginning of the nineties did not exist any toolkit that supported the development of graphical interfaces for desktop computers and a lot of effort was dedicated to the coding of the user interface. Precisely, Myers and Rosson [1992] found that approximatively the 48% of the code was committed to the programming of the user interface. That was the case before the development of programming toolkits. For example, Myers and Rosson [1992] demonstrated that GUI development kits, such as MacApp, could reduce the development time by at least a factor of four. In fact, thanks to GUI toolkits, developers need less code for the user interface that, therefore, can be generated more quickly. This assists the rapid prototyping, which is a pivotal activity to achieve high quality

interactive products through iterative design [Lim et al., 2008]. Following the same rationale, toolkits for ubiquitous interaction might help designers in the development of novel systems, especially in a context where not only the software interface matters, but also the hardware component is important. To this regard, Myers et al. [2000, p.18] pointed out that:

An important consideration for the new devices is unlike desktop machines that all have the same input and output capabilities (mouse, keyboard, and color screen), there will be a great variety of shapes, sizes, and input-output designs. Much of the user interface will be built into the hardware itself, such as the physical buttons and switches. Therefore, the designer will have to take into account not only the software, but also the physical properties of the devices and their capabilities.

Prototypes have a fundamental role in HCI and design: they can be used to evaluate a design in its early stages, but also to foster innovation and creativity, by enabling the exploration of a design space [Lim et al., 2008]. Nevertheless, while prototyping tools are common for classical interaction with GUIs, prototyping interaction for ubiquitous systems is still an issue [Wu et al., 2012]. The tremendous amount of different input devices, interaction techniques and physical environments envisioned by researchers makes it difficult to work in this technological context and, at present time, only a few tools support this type of development. Moreover the design and development space is further complicated by the fact that *"many of the sensing and display technologies used by ubiquitous interfaces are still in the process of evolution and maturation, and have not yet reached the state of stability and robustness"* [Wu et al., 2011]. The need in the art to have better prototyping tools motivates this dissertation. Tools that support the rapid setup of ubiquitous environments reduce the effort in arranging the technological medium and, as a result, allow researchers to focus on social aspects and identify patterns and emergent trends in collaborative work with heterogeneous devices. For example, *"little is known about how best to design or evaluate such 'device ecologies'; in particular, how best to combine devices to achieve a desired type of collaborative user experience"* [Coughlan et al., 2012]. Collaborative activities among people with different backgrounds and areas of expertise are crucial in our times and, in order for interactive technologies to support them, not only new models of interaction in ubiquitous environments are needed, but also effective tools to prototype device ecologies, having the potential to provide time on task by lowering prerequisite knowledge and by automating *'low-level'* programming skills [Shneiderman, 2007]. This dissertation directly addresses the problems related to the development of prototyping tools for interactions between heterogeneous devices.

1.4 Methodology

After twenty years of original Weiser's vision, even if ubiquitous interfaces are in a continuous evolution, fundamental enabling technologies and interaction paradigms have been established and there are the basis to apply a more design-oriented approach to the research area of Ubiquitous Interaction. In this research work, the Design and Development Research approach has been followed, defined by Hasan [2003, p. 7] as a *"disciplined investigation conducted in the context of the development of a product or program for the purpose of improving either the thing being developed or the developer"*. First of all, since part of this work is about the development of a conceptual framework and its manifestation as a software library, it is compelling to highlight the differences between *'Design and Development Research'* and *'product development'*. According to Ellis and Levy [2010], in order to be considered research, the development of an artifact has to:

1. **Address a problem recognized by the research community.** In general, problems in design and development research are complex and multidimensional and the development of the artifact has to potentially allow to address such problems through a form of human creativity or interaction. By contrary, the development of a product is often driven by direct, straightforward and unidimensional problems that usually do not motivate research.
2. **Rely upon existing literature.** To make a successful contribution it is important to start from the existing corpus of knowledge, placing the research in the context of the state of the art.
3. **Make an original contribution to the corpus of knowledge.** An original contribution in the case of design and development research can present different characteristics. First of all, according to Hevner et al. [2004], it has to be useful; a target group of users has to be able to employ the artifact to address the problems at issue. Contributions also take the form of: (a) systematic documentation of the process that includes a discussion of design choices made, options considered, and rationale for the alternative selected [van den Akker, 2000], (b) empirical testing of the artifact developed [Hevner et al., 2004] and, (c) communication of the results of the process [Hevner et al., 2004].

Due to the literature in *Design Science in Information Systems* research [Ellis and Levy, 2010; Hevner et al., 2004] and in particular to Peffers et al. [2007], it is possible to define a six-phase conceptual framework that guides Design and Development Research. Figure 1.4 shows the six phases, which are: (a) identify the problem motivating the research; (b) describe the objectives; (c) design and develop the artifact; (d) subject the artifact to testing; (e) evaluate the results of testing; and (f) communicate those

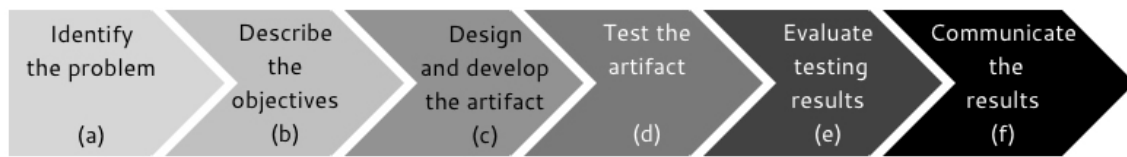


Figure 1.4: The six-steps design and development research approach.

results. In this work, the original linear model has been adapted to an iterative process, as shown in Figure 1.5, in which the output of each step may also serve to refine or redefine the outcomes of previous steps. In the next Sections details are given about the activities performed for each stage of the proposed model. For the sake of simplicity and understanding, the activities for the two stages *d* and *e* have been grouped into a single Section.

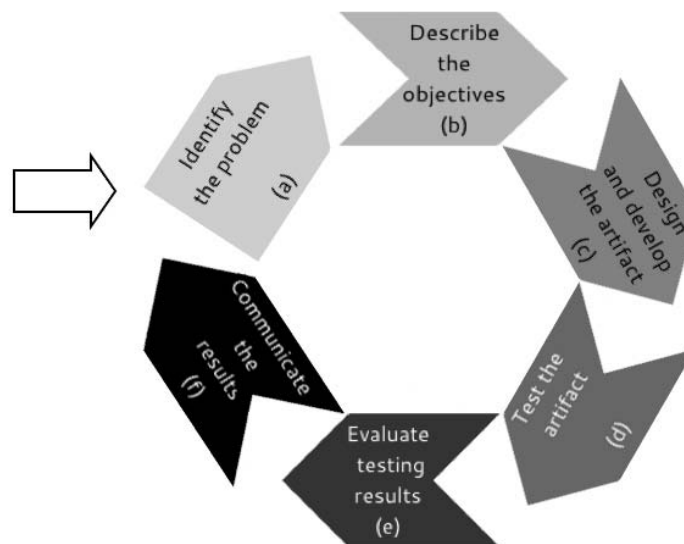


Figure 1.5: The modified six-steps design and development research approach.

1.4.1 Problem

The identification of the problem motivating the dissertation has been conducted through the careful inspection of the existing literature in the HCI, UbiComp and IxD. Personal knowledge has also been exploited, gathered during the development of multi-touch, tangible and touchless interactive systems [Bellucci et al., 2011, 2010]. Focusing on the technological side, it has been found that the current infrastructure for Ubiquitous Interaction is not homogeneous or seamless, but it is fragmented into individual devices, software and interaction techniques that the user has to understand and learn how to use. There are devices for augmented reality, for the recognition of full body movement,

microcontrollers for physical computing, software libraries for the recognition and tracking of touch input, protocols such as Tangible User Interface Objects (TUIO) [Kaltenbrunner et al., 2005] for the definition of touch messages, libraries that implements computer vision algorithms, tags for tangible objects, touchless interfaces and so on. Employing such variety of hardware devices and software environments in a real application can be difficult because their use requires specific knowledge of the hardware components and many hours of programming work. Furthermore, data integration can be cumbersome, for any device vendor uses different software interfaces and communication protocols. All of this results in a messiness that makes the development of Ubiquitous Interaction a challenging task.

The main problems that have been identified, which will be exposed in details in the Chapter 3 are: (1) the lack of holistic tools and (2) the consequent elevated temporal and technical requirements for the prototyping of physical artifacts with digital behaviors. Refer to Section 2.6 for a discussion on the role of prototyping in Ubiquitous Interaction. As Hevner et al. [2004] stated, *"Design and Development research would not be appropriate if the development of some form of artifact does not present any potential for addressing the problem"*. In this case, the design and development of a comprehensive framework is motivated by the lack in the literature of unified approaches and it specifically addresses the needs of developers and programmers to ease the prototyping of ubiquitous interaction. Having an agile tool to rapidly setup technologically-enhanced spaces, in turn, allows to focus research efforts on the human side: *how the interaction with these devices affect human activities and collaboration?* For the sake of clarity, human-related challenges are not addressed by this dissertation, which aims at developing the technological support for the rapid setup of device ecologies.

1.4.2 Objectives

The main objective of this work is **to define a conceptual model and a reference architecture that provide users with a comprehensive framework for managing interactions between heterogeneous networked devices to support the rapid prototyping of ubiquitous interactive systems**. The hypothesis that is demonstrated in this dissertation is that **by encompassing heterogeneous devices into a unique design it is possible to reduce user efforts to develop for interactions in ubiquitous environments**. The goal is to allow the development of ubiquitous interactive systems within the structure of a single, unified and holistic reference architecture. The framework offers a general architecture that define communication protocols to describe interactions between and with ubiquitous devices. The materialization of the framework led to the definition of a set of software libraries that can be directly used by developers in their projects. The framework, therefore, is aimed at fostering rapid prototyping by explicitly addressing the two research issues summarized in the Section 1.4.1.

1.4.2.1 Contributions

On the road to fulfill the goals proposed by this work, the following contributions have been produced:

1. **The definition of a set of requirements for the framework.** The first step of an engineering project is the specification of the requirements with respect to the demands of its various stakeholders. In this case, the requirements of a framework that will be used to develop prototypes for ubiquitous interactive systems have been highlighted.
2. **The development of a general and modular hierarchy, based on primitive data types,** where the top-level components allow for flexible and generic access to device features. Abstraction has been pursued from the low-level details of specific devices. In this way it is possible to provide unified access to devices, independently of their implementation or communication protocols. To this end, a data types structure has been defined for communication between devices in an ubiquitous environment. Abstracting from heterogeneous devices implementations requires the definition of a high-level data types structure that can describe raw data from hardware devices in a unified manner.
3. **The development of a software library for ubiquitous interaction** that employs hardware abstraction to ease the prototyping process. Providing access to devices, sensors and emitters by means of a unified, high-level API results in the support of the rapid prototyping of interactive systems and the reuse of software components in different applications so to reduce their development time and make it possible for developers to quickly explore numerous designs.
4. **A case study** of an interactive system that demonstrate how the software framework can be used to implement actual applications.
5. **The demonstration,** through a user study, that a unifying framework that make use of hardware abstraction positively affects the efficiency in the prototyping of interaction with device ecologies.

1.4.3 Designing and Developing the Artifact

At this stage, the waterfall development method [Pressman, 2001] has been followed, taking especially into account three main factors: (1) the definition a conceptual framework, (2) the design of the system architecture, and finally, (3) the development of a prototype for testing and evaluation.

The first factor corresponds to the definition of a set of requirements as highlighted in the first contribution of the thesis (see Section 1.4.2). The definition of requirements is crucial because they connect the software artifact being developed with the problem driving the development. In this particular case, such requirements will connect the lack of a general framework that integrates support to heterogeneous devices with the actual conceptual architecture. They have been extracted from the review of the literature, from the experience of developers and researchers familiar with ubiquitous technologies (both from the academia and the industry) and, eventually, from my personal experience. With respect to the second factor, the decisions made to design the architecture of the framework are described in this document, discussing the rationale behind the choices and the alternatives taken into account. In particular, the semantic of virtual devices by Wallace [1976] has been explored and exploited, in a similar way as for the Squidy library [König et al., 2009], to establish a general data types hierarchy for ubiquitous systems. Finally, the software framework has been developed, which exposes a software library to interface with a limited set of devices, sensors and actuators. This prototype has been used during the next stage of testing and evaluation.

1.4.4 Testing and Evaluating

In this last phase, the proposed approach has been evaluated within the conceptual scheme for user interface development defined by Myers et al. [2000]. The evaluation focused on the five themes they established, namely:

- **The parts of the user interface that are addressed:** The tools that succeeded helped (just) where they were needed.
- **Threshold and Ceiling:** The “threshold” is how difficult it is to learn how to use the system, and the “ceiling” is how much can be done using the system. The most successful current systems seem to be either low-threshold and low-ceiling, or high threshold and high ceiling. However, it remains an important challenge to find ways to achieve the highly desirable outcome of systems with both a low threshold and a high ceiling at the same time.
- **Path of Least Resistance:** Tools influence the kinds of user interfaces that can be created. Successful tools use this to their advantage, leading implementers towards doing the right things, and away from doing the wrong things.
- **Predictability:** Tools which use automatic techniques that are sometimes unpredictable have been poorly received by programmers.
- **Moving Targets:** It is difficult to build tools without having significant experience with, and understanding of, the tasks they support. However, the rapid development

of new interface technology, and new interface techniques, can make it difficult for tools to keep pace. By the time a new user interface implementation task is understood well enough to produce good tools, the task may have become less important, or even obsolete.

The following evaluation scheme with respect to the five themes has been used:

- **Evaluation task 1.** An approach based on use cases has been employed, through which it has been possible to demonstrate that the proposed framework is powerful enough to model and implement real ubiquitous systems. As testbed it has been used a multimodal application developed in the DEI Laboratory at Universidad Carlos III de Madrid using the proposed software framework. The evaluation gives answers to the following questions:
 - *The parts of the user interface that are addressed:* does the framework succeed in implementing the application? What kind of real applications can be developed with the framework?
 - *Path of Least Resistance:* is it possible to create proper interaction for different kind of input devices without changing the underlying infrastructure?
 - *Moving Targets:* how does changing the input device affects the development?
- **Evaluation task 2.** An empirical evaluation of the software libraries has been conducted with a group of users with programming knowledge. The group of users has been carefully selected considering the profile of the target population of the framework. The main objective of the test was to collect quantitative and qualitative data to evaluate the usability of the solution. With this evaluation it has been possible to find out whether the framework is able to adhere to:
 - *Threshold and Ceiling:* how difficult was to learn how to use the APIs? Which kind of interaction can be developed using the APIs?
 - *Predictability:* does the API behave as expected by the developers?
 - *Moving Targets:* how does changing the input device affects the development?

1.4.5 Communicating the Results

The results of the evaluation, in addition to the document for this thesis dissertation, will be distributed in contributions to workshops and conferences, such as the Physicality Workshop⁵ or the Conference on Human Factors in Computing Systems⁶ (CHI). Appendix A lists the scientific contributions resulted from this research that have been already published or accepted.

⁵<http://www.physicality.org/Physicality.org.html>

⁶<http://chi2013.acm.org/>

1.5 Outline

In Chapter 2 this thesis work is placed in the context of the state of the art of libraries, toolkits and frameworks that make it possible the prototyping of ubiquitous interaction. In Chapter 3 the research issues addressed by this dissertation are presented. These issues have been defined from the state of the art, interviews with developers and HCI researches and my personal experience (as a developer and researcher). In Chapter 4, the design space of a framework for Ubiquitous Interaction is framed, reporting on the experience and the lessons learnt from the development of ubiquitous systems. In this same chapter, the requirements for the framework, together with details on the process for their elicitation, are discussed. In Chapter 5 the development of the conceptual and software framework is presented, highlighting on the interaction model it follows and its peculiarity with respect to other solutions in the state of the art. Implementation and technical details on the software library and the communication protocol are also addressed. Testing and evaluation are the subject of Chapter 6: the use case and the user study are described here. In particular this Chapter reports on the details of the testing setup, to what extent the use case proved to fulfill the objective and the analysis of experimental data from the user study. Conclusions are drawn in Chapter 7 and also potential for future research is presented.

2

State of the Art

And if everything have already existed, what think you, dwarf, of This Moment? Must not this gateway also - have already existed? And are not all things closely bound together in such wise that This Moment draws all coming things after it? consequently - itself also?

—*Thus spoke Zarathustra*

FRIEDRICH NIETZSCHE, GERMAN PHILOSOPHER

IN this chapter the state of art of the UbiComp paradigm is framed under the perspective of Human-Computer Interaction, from which the definition of Ubiquitous Interaction (see Section 1.2) is derived. The chapter offers an overview of the concept of device ecologies and its manifestation in terms of technologically-enhanced spaces and then, it focuses on different technologies and techniques that enable interaction in these kind of physical environments enhanced with computational capabilities. Major interaction technologies in UbiComp are presented, such as: touchless/remote, multi-touch and tangible. The discussion of interaction technologies is completed by introducing the state of the art of software libraries, toolkits and frameworks support to the development of technologically-enhanced spaces. Since the framework proposed in this dissertation encourages the exploration of the design space of Ubiquitous Interaction, the concept of rapid prototyping in HCI is also discussed here, with a particular emphasis on the rapid prototyping of ubiquitous systems. Interaction in such systems requires hardware and software components: for this reason, in the art, there are software toolkits that ease the development of hardware components, such as Phidgets [Greenberg and Fitchett, 2001] or Arduino [Mellis et al., 2007] and software libraries that builds on existing hardware, such as Mt4j [Laufs et al., 2010], Papier-Mache [Klemmer et al., 2004] and ROSS [Wu et al., 2012]. The proposed solution focus on the software side, aiming at creating a software infrastructure that ease the development of interaction with a large variety of different devices.

2.1 Device Ecologies

An important issue in UbiComp is to explore novel forms of interaction not just between a person and a single device, but also (a) between heterogeneous devices of different form factors and capabilities, (b) between an individual, his personal devices and the devices that harmonizes with the surrounding environment and (c) between different persons through the use of different devices and interaction techniques. As also discussed in Section 1.1, the UbiComp scenario presents socio-technical issues when it comes to interaction—this work focuses on the technical aspect of device ecologies and, in particular, it addresses the point (a) and (b) by providing a common architecture that allows communications between heterogeneous devices and user interactions with devices of the surrounding environments in a comprehensive framework. Weiser [1991] argued that the real power of ubiquitous computing *"comes not from any one of these device—it merges from the interaction of all of them"*. With the widespread adoption of new technologies that allow computation to go beyond the traditional desktop settings, the vision of Weiser is approaching to its materialization, even if the user experience is still disordered, as Oulasvirta [2008] (see Section 1.3 for details) and other researchers such as Rogers [2006] and [Dourish and Bell, 2011] pointed out. This lack of organization is due, in part, to the fact that devices are designed in a relatively isolated way and then put in the same environment with other devices, without considering *"how to configure the varied properties of devices into holistically-designed system, with desirable characteristics in terms of the assemblage of people, artifacts and technologies, and the transitions that occur between them"* [Coughlan et al., 2012]. To cope with the messiness that characterized current ubiquitous environments it is important to understand how heterogeneous technologies can be organized into an *ecology*, with different interface, devices, tangible objects, physical and digital data and interconnections between artifacts and people. A *device ecology* has been defined by Loke and Ling [2004] as—*"collections of devices interacting synergistically with one another, with users, and with the Internet"*. As exposed in Section 1.1, a practical materialization of this concept are technologically-enhanced spaces where various touchless, multi-touch and tangible input devices of different form factors coexist in the same environment (e.g., a meeting room or a *command & control* room) as a support to collaborative activities between people. In general, the typical setting envisages the presence of large to medium-sized public display screens and private devices [Streitz et al., 1999]: how to best combine shared and personal devices is a principal research topic. Apart from the i-LAND project, mentioned in Section 1.1, other examples of technologically-enhanced spaces, where shared display coexists with personal devices and a variety of interaction techniques are adopted, are WeSpace [Wigdor et al., 2009] and CodeSpace [Bragdon et al., 2011]. WeSpace was designed for collaborative scientific discussion, with a shared tabletop interface controlling a large vertical display. It also allowed the scientists to integrate their laptops and share their own data on the large display. It was found that individual users contributed equally,

suggesting this form of design is suited to equitable access and interaction. CodeSpace

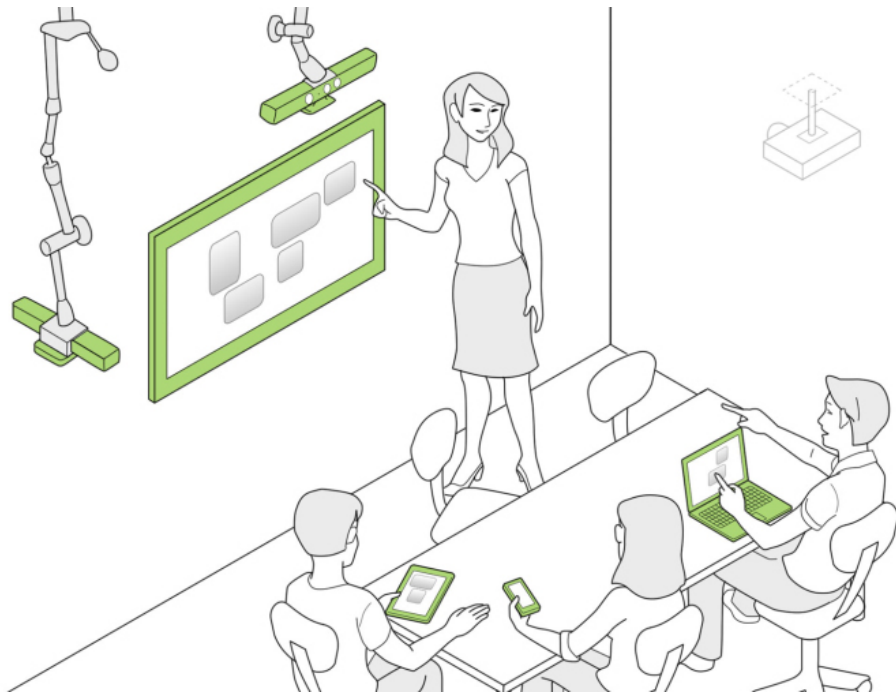


Figure 2.1: A sketch of CodeSpace meeting environment. [Bragdon et al., 2011]

(see Figure 2.1) uses a combination of a shared multi-touch screen, mobile touch devices, and Microsoft Kinect sensors to support meetings of a small group of developers. They implemented cross-device interactions via different techniques such as in-air pointing for social disclosure of commands, targeting and mode setting, and multi-touch for command execution and precise gestures. The CodeSpace project is therefore a clear example of the diversity of components that constitutes a device ecology: interactions seamlessly span modalities and devices, forming a cross-device architecture where everyone can interact with the shared display, from anywhere in the meeting space, with any device they bring. A potential problem of device ecologies is that the users might feel frustrated by the wide variety of interactive options offered by different input devices. Cross-device and multi-user interfaces, for example, involve a whole new set of cues about objects to which users direct their actions or intentions. Indeed, it has been found that sometimes people cannot keep track of what is going on and will ask their partners to 'wait' or 'slow down' [Harris et al., 2009]. A key challenge, therefore, is determining how to exploit the characteristics of individual devices and how these can be combined to best effect to support various collaborative activities.

2.1.1 Physical Computing

The term *Physical Computing* [Igoe and O'Sullivan, 2004] is used to describe human-computer interaction that occurs using a physical medium. A physical computing system is composed by software and hardware that can sense the surroundings and respond to physical stimuli with both digital and physical outputs. To fulfill this goal, a physical computing system communicates with the real world by means of sensors and emitters. Sensors convert real world inputs into digital data, while emitters are mostly used to provide digital or physical feedback. In general, these are embedded systems that use very simple computers called microcontrollers that read sensors input and convert into data to be processed. Physical Computing can be viewed as a conceptual framework to understand ubiquitous interaction because it primarily deals with physical interaction with everyday objects enhanced with computational capabilities. In practice, the term is widely adopted to describe design or DIY projects that use sensors and microcontrollers to translate analog input to a software system, and/or control electro-mechanical devices such as motors, servos, lighting or other hardware. Physical computing activities are widely present in the interaction design domain, where design practices are supported by the use of embedded systems to create rapid prototypes of physical/digital artifacts that can be used as cost-efficient concepts throughout the design process. Examples of physical computing practices can be found among well-known product design firms such as IDEO¹ and Teague².

2.2 Touchless/Remote Interaction Technologies

As an interaction technique, *touchless* or *remote* refers to an input that is originated without any physical contact with a surface, in contrast with touch interactions that suggests the presence of an input device that hits the surface, like for example a stylus or users' hands. By allowing users to employ hand gestures, touchless input removes the burden related to physical contact and promote *natural* interaction with digital information made tangible through large display surfaces [de la Barré et al., 2009]. After many decades of research, the ability to interact with technology through remote hand gestures and body movements is becoming an everyday reality. The emergence of cheap technologies based on sensing cameras and computer-vision approaches, such as Microsoft Kinect³ among a host of other related technologies, has had a profound effect on the collective imagination, inspiring and creating new interactions that go beyond traditional input mechanisms [O'Hara et al., 2012]. First examples of touchless interaction can be found in the work of Bolt [1980], Krueger et al. [1985] and Baudel and Beaudouin-Lafon [1993].

¹<http://www.ideo.com>

²<http://www.teague.com>

³<http://www.microsoft.com/en-us/kinectforwindows/>



Figure 2.2: Put-That-There [Bolt, 1980].

In 1980, Richard A. Bolt from MIT presented *Put-That-There* (Figure 2.2), a multimodal interface for managing spatial information displayed on a large-screen display surface. It makes use of a sign and speech paradigm in which users execute specific commands via hand-gestures and voice. The touchless interaction style, combined in a multimodal interface [Oviatt, 2003], allows the system to support multiple users interacting at the same time: no single person blocks the screen or has absolute control of the system via a mouse and keyboard interface. Conversely from Bolt, Krueger et al. [1985] developed VIDEOPLACE, a VR environments that was not based on the mapping between physical gestures and digital actions. They used, instead, direct free-hand gestures to manipulate projected images: the user interacts with a virtual world using the same gestures they are used to employ in the real world. After nearly a decade, Baudel and Beaudouin-Lafon [1993] presented CHARADE, another touchless gesture-based system which (a) defines active zones on the display surfaces to distinguish between gestures used to execute a command from other gestures and recognizes dynamic gestures. The analysis of these three systems reveals some of the main issues of early hand-gesture interaction, which still have implications on the design of touchless interaction nowadays:

- The dichotomy between *natural* and *artificial* gestures. In *Put-that-there* and *CHARADE* users are forced to learn a gesture or speech vocabulary defined a priori and, for this reason, Baudel and Beaudouin-Lafon [1993] recommended that “*gestural commands should be simple, natural, and consistent*”. By contrary, in

VIDEOPLACE's gestures mimic real world actions.

- At that time, gloves or other physical sensing devices worn by the user and linked to the computer were needed to send gestural input to the system. As Fukumoto et al. [1992] foresaw, capturing gestures by using external cameras or optical sensors and computer vision techniques would have overcome the problem.
- These kinds of systems are really sensitives to user's hand motion. As a consequence, they interpret every gesture whether or not it was directed to the system. Therefore, while interacting with such systems, the user cannot communicate simultaneously with other devices or people.

Gestures based interactions gained renewed interest during the last ten years in the Natural User Interface (NUI) community and reach the maturity to establish as a common input technique, especially when interacting with large screens [de la Barré et al., 2009]. Remember the movie *Minority Report*⁴ and the touchless interface Tom Cruise was working with for managing visual objects on a semitransparent curved screen surface (see Figure 2.3). At the time you saw the movie, you probably considered this



Figure 2.3: *Minority Report*: envisioning a touch-free interface.

kind of interaction only belong to science fiction and that we would never have seen it in the real world. Nevertheless, due to advances in technologies and HCI research, many prototypes of *Minority Report*-style interfaces have been proposed, in the hope that they will be affordable and widespread in the years to come. Several research

⁴<http://www.imdb.com/title/tt0181689/>

efforts have been recently focused in enabling human-display motion-based interactions (e.g., detection of natural body movement). Most of the emergent devices comes from the entertainment industry, such as: Nintendo's Wii Remote Controller (also known as Wiimote) or Microsoft Kinect. Besides of providing a more intuitive game playing, not only based on button pushing [Vaughan-Nichols, 2009], these devices also stimulate the development of touchless interfaces that go beyond classical Windows, Icons, Menus and Pointer (WIMP) interaction styles. The Nintendo Wiimote (Figure 2.4) is one of the



Figure 2.4: Nintendo Wii Remote Controller.

most popular remote input devices. It was also one of the most sophisticated when it first appeared, providing a variety of multimodal I/O functionalities. The Wiimote is mostly advertised for its motion-sensing capabilities: users can interact with a computer system via gesture recognition or pointing by exploiting the built-in accelerometer and the InfraRed (IR) camera tracker. The most interesting part of the controller is its IR camera, which provides an image-processing engine that can track up to four moving objects and send their coordinates to a host, giving users fast and high-precision tracking at a very low cost. The Wiimote arrival was a first step towards the widespread availability of gesture-based interfaces. It provides a cheap and effective solution to develop touchless applications at home or in the laboratory without requiring much more than employing adequate APIs, building small LED-based devices and basic programming skills [Bellucci et al., 2010]. Integrating the Wiimote with the surrounding environment is thus straightforward and can

promote the use of ubiquitous applications even when engineering and economic factors are an issue. In fact, it is possible to develop a solution such as a low-cost multipoint interactive whiteboard using the Wiimote with a relatively small budget, compared to off-the-shelf solutions [Lee, 2008].

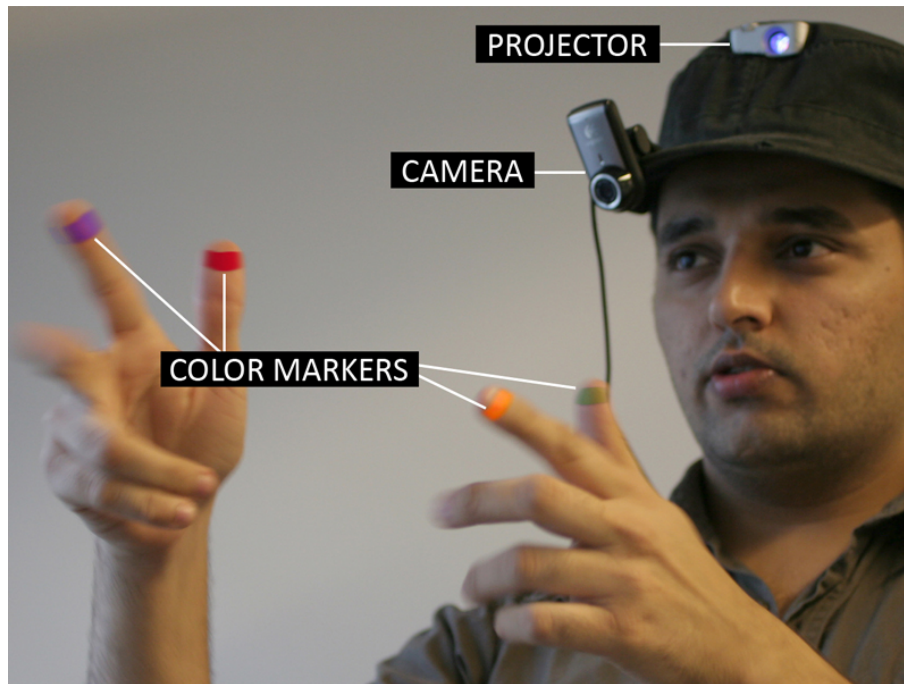


Figure 2.5: SixthSense technology developed at Fluid Interfaces Group, MIT Media Lab [Mistry and Maes, 2009].

The use of hand gestures to interact with digital information displayed in the real world (see Figure 2.5) has been demonstrated by the SixthSense hardware [Mistry and Maes, 2009], a wearable gestural interface that is composed, at its core, by a pocket projector, a mirror, and a camera. The projector turns tangible surfaces—such as walls and physical objects—into screen displays by projecting visual information. A software processes the camera's data and tracks colored markers on the tips of the user's fingers using naive computer vision techniques. The G-speak spatial operating environment (developed by Oblong Industries)⁵ is one example of a more sophisticated Minority Report-like environment (Figure 2.6). It is a touch-independent interactive environment that allows user to interact with projected screens and tabletop surfaces by using freehand gestural inputs. Due to its precise accuracy in recognizing the gestural input it shows the potentiality of using these techniques in a pervasive environment, where users are surrounded by display surfaces of different kind.

⁵<http://oblong.com>



Figure 2.6: G-Speak interactive environment from Oblong Industries.

The idea of *body-as-sensor* is the main feature of Microsoft Kinect. It was born as a “*controller-free gaming and entertainment experience*” for the Xbox 360 video game platform and now it has been converted into a leading technology to enable touchless interfaces by letting users interact through full body movements. The device features a Red Gree Blue (RGB) camera, a depth sensor, a multi-array microphone and a custom processor running proprietary software to provide full-body 3D motion capture, facial recognition, and voice recognition capabilities. Affordable RGBD cameras (a system made of a normal RGB camera plus a depth sensor), with real-time synchronized color and dense depth like the one provided by the Microsoft Kinect, are improving and fundamentally change the way computers can perceive a real world scene. In fact, thanks to computer vision algorithms, users do not need to wear special gloves or hold input devices in order for the system to recognize their movements. These new hardware and software solutions are therefore giving tremendous chances to implement interactive systems that are more invisible and integrated with the environment, thus providing improvements in the road to the materialization of a real ubiquitous interaction. Optical solutions for touch free interaction existed long before PrimeSense⁶ developed the sensor behind the Kinect success. The real advantages of the Kinect are:

- the development environment, which provide programmers with integrated tools to make sense of the data retrieved from the camera (e.g., middlewares for gestures recognition and tracking) and,
- the low price. In fact, if compared with early Time-of-Flight (ToF) cameras, which is a different technique to generate distance matrix of the environment, the cost of the Kinect is up to 10 times lower. While in general terms cameras based on

⁶<http://www.primesense.org>

ToF technique has a better cover range and speed (up to 60m and 100fps against the 10m and 30fps of the Kinect), this level of precision is not needed for enabling touchless interaction, which makes it possible to drop the price. The result has been that, on the wake of the Kinect hardware, novel and affordable ToF cameras has been produced with a specific orientation to human-computer interaction, such as the PMD Camboard Nano⁷.

2.2.1 Implications for Interaction Design

Touchless input introduced in the HCI research the narrative of *natural* interaction as a way to communicate with a computational system by adopting the same gestures humans would use in real-life tasks. Some authors suggest that a natural design implies a direct matching between *"the behaviour of the system to the gesture humans might actually do to enable that behaviour"* [Saffer, 2008]. This same vision of natural interaction also includes multi-touch, tangible and natural language input technologies. However some major concerns has been raised with respect to the equivalence of '*natural*' and '*intuitive*': many researchers [O'Hara et al., 2012; Wigdor and Wixon, 2011] argue that allowing people to use common gestures of everyday life not necessarily results in an optimal interface design. For example, Norman [2010] critiqued the naturalness of gestural interfaces in terms of their claimed intuitiveness, usability, learnability and ergonomics, Wigdor and Wixon [2011] pointed out that naturalness is a design goal that makes interaction an enjoyable, effortless and graceful experience and, just to cite a last one, O'Hara et al. [2012] argued that current works focus on naturalness only in term of interface but, in order to deeply understand opportunities and constraints we need to be *"less focused on the in situ and embodied aspects of interaction with such technologies"*.

On the other hand, the touchless input devices presented above demonstrated how the technology has reached a sufficient level of maturity to allow the development of ubiquitous environments that make use of remote interaction. With current technology it is possible to develop prototypes for including touchless gestures in the interaction with distant screen displays, thus allowing researchers to evaluate the use of such input modalities in a collaborative context. It is crucial to understand the role of touchless input in human activities and how the task might influence the choice of the input device. There is no input technique that best suits for every task and, therefore, the issue is to understand whether hand-free input can be a good solution for certain tasks, or the use of a physical device can be a better option. Moreover, there can be cases where there is no difference in employing or not a physical prop. For example, Freeman et al. [2012] demonstrated that there are no significant changes in using free-hand, a stick (low fidelity) or a paddle (high fidelity) controller in a tennis table video game. This example highlights how there is a need for tools that allow rapid prototyping of touchless

⁷<https://www.cayim.com/index>

interaction, making it possible to study several remote input devices even at early stages of design to gain a better understanding of how we experience the world in new ways, accordingly to the technology in use [O'Hara et al., 2012].

2.3 Multi-touch Interaction Technologies

The term *multi-touch* refers to a technology that allows users to interact with more than one finger touch at time, taking into account finger movements as well as hand gestures. Devices that are capable of detecting touch are often classified colloquially as either *touch* or *multi-touch*, but there are distinctions that help to define the terminology of touch-based interaction. Single-touch devices, such as traditional resistive touch screens, are adequate for discrete input or for emulating a mouse [Potter et al., 1988]. Single-touch means that the detection and control of touch events is made over a single contact point: that is, single-touch systems respond to the movements of only one finger or stylus, like for instance the first Portable Digital Assistants (PDA). In multi-touch, two or more touch events can be detected at the same time and movements of contact points are tracked individually. The number of contacts points they can handle is an important feature of multi-touch devices. Two-touch systems are able to recognize and track two contact points and they enable gesturing (e.g., two fingers pinch-to-zoom on the Apple iPhone). Multiple contacts (more than two) are required for users to perform multi-touch gestures such as multi-fingered grabbing [Moscovich and Hughes, 2006]. Still more contacts must be tracked to enable multiple users to preform multi-touch gestures at once, as desired for collaborative tabletop interfaces [Dietz and Leigh, 2001]. In general in a multi-touch environment, the term *gesture* does not relate to the expressive gestures used in human face-to-face communication but instead to familiar and conventional hand movements used in some particular task (turning the hand with some fingers pressed on the surface can be interpreted as a gesture for rotation). A description of a multi-touch gesture includes attributes such as position, motion velocity and acceleration of each contact point that the software will use to recognize the gesture and activate the associated interface action: for example, the de facto standard, two fingers pinch-to-zoom gesture or five fingers pinch (grab) to exit an application on the iPad.

Since our world is shaped by the presence of physical surfaces, the use of multi-touch interactive screen displays has gained interest in ubiquitous interaction for a wide range of everyday life settings. As Rekimoto and Matsushita [1997] pointed out—*“It is difficult to imagine offices, museums or homes without walls. Thus it should be worthwhile to research how computer augmented walls will support our daily activities”*. Walls are just an example: a wide variety of horizontal surfaces like tables, desktops, floors and even the surfaces that shape common objects coexist together with such large vertical areas. The raising interests in digitally support human interactions with surfaces motivated the need to augment their physical affordances with computational power. Since the

very beginning, the cost and technological complexity for designing and developing such systems, where the physical meets the digital, have represented a barrier to the diffusion of interactive surfaces. Nevertheless, recent hardware developments have caused the widespread proliferation of interactive display technologies, together with novel human-display interaction possibilities, such as multi-touch input. For example, we can think about the famous demonstration at TED⁸ by Jeff Han, who exploited the Frustrated Total Internal Reflection (FTIR) optical phenomenon to build a low cost interactive tabletop [Han, 2005]. This technology is based on the optical phenomenon of Total Internal Reflection (TIR). The light incident on the space of boundary (the interface) between two materials, with different refractive indices, is transmitted to the second medium and partially reflected back to the first medium. Total Internal Reflection takes place when the angle of incidence of the ray of light exceeds a *critical angle* $\theta = \arcsin(\frac{n_1}{n_2})$, with n_2 the refractive index of the less optically dense medium, and n_1 the refractive index of the more optically dense medium. Han's method uses this optical effect to trap infrared light within a piece of acrylic. When the user's finger comes into contact with the surface, the light rays are said to be frustrated, since they can now pass through into the contact material (skin): the reflection is no longer total at that point and the light is scattered outside the acrylic (Figure 2.7).

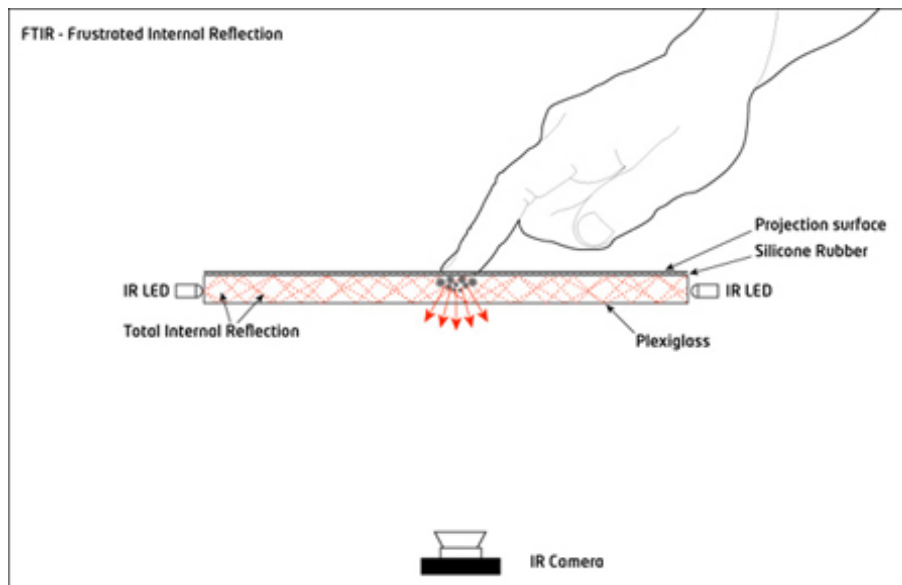


Figure 2.7: FTIR configuration for multi-touch displays.

From the industry, the use of capacitive sensing screens resulted in the adoption of touch-enabled displays for mobile devices like smartphones and tablets: well-known examples are the iPhone and the iPad, which removed physical buttons in favor of a full display interface operated with touch gestures. In order to integrate multi-touch and

⁸<http://goo.gl/xlOCv>

tangible sensing in ubiquitous environments, the growth of hardware solutions has come along with the development and use of software libraries that made easier to program interaction. As happened for the hardware, Application Programming Interfaces (APIs) are intended to lower the complexity of building touch-enabled interactive systems: they aim at relieving developers from low-level implementation details, thus making it possible to focus their creative efforts on interaction.

Research in (multi-)touch interaction is about 25 years old [Buxton, 2010]. In 1987, Apple produced a video presenting the Knowledge Navigator⁹ concept prototype, as described by Sculley [1987] in his book *Odyssey*. Knowledge Navigator was essentially a device that can access a large networked database of hypertext information, and use software agents to assist searching for information. The device had a tablet form factor with a two-page book layout. The user interacted with the system via a multimodal interface, by touching the display or simply by using natural language commands. Although this vision represented an unrealistic depiction of the capacities of any software agent, even in a distant future, Knowledge Navigator concept was the precursor of multitouch interfaces for portable devices, proposing a touch-based interface looking like the multitouch interface later used on the Apple's iPhone¹⁰. Augmented Surfaces [Rekimoto and Saitoh, 1999] was the first project to integrate touch input in an environment where users can exchange digital information by employing laptop computers, tabletops, wall projected displays and physical objects (Figure 2.8).

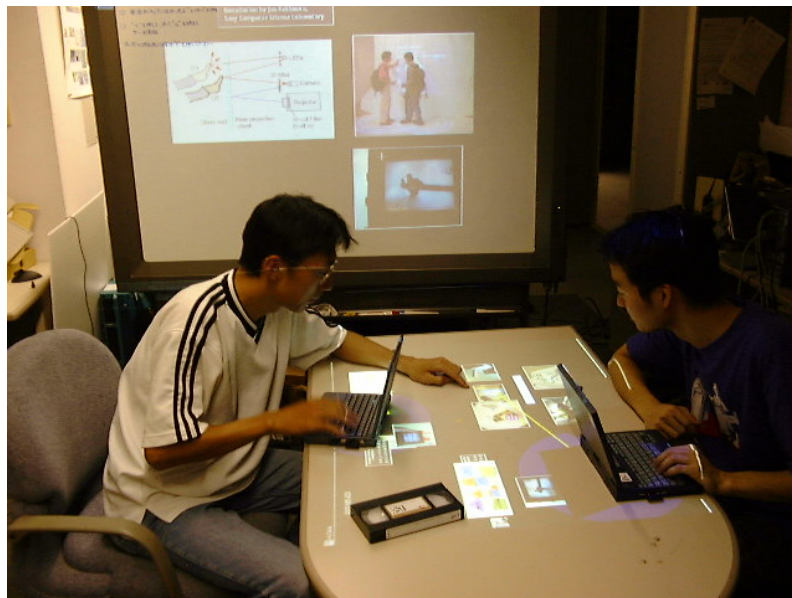


Figure 2.8: Augmented Surfaces [Rekimoto and Saitoh, 1999].

The system was designed following ubiquitous computing principles: users can

⁹http://youtu.be/QRH8eimU_20

¹⁰<http://www.apple.com/iphone/>

exchange digital information by employing laptop computers, tabletops, wall projected displays and physical objects. Therefore, Augmented Surfaces aim at the integration of mobile devices in a digitally augmented collaborative space, consisting of an interactive table and wall, where participants use their laptop computer to interact. A video camera mounted above the table recognizes the device by means of an attached visual marker: as a result of the recognition, the table surface becomes an extended workplace for each laptop computer. For example, Augmented Surfaces implements a technique called hyperdragging that allows users to seamlessly drag an object across the boundaries of the computer display. When the cursor moves on the table surface, a line is projected to show a connection between the cursor and the laptop. Users can also move data from the table to the wall surface using a laser pointer. Tagged physical objects are exploited to create bindings between the real and the virtual world: for instance, users can drag digital pictures to a physical booklet placed on the surface. In this way, the booklet becomes a real storage for virtual images, which can be retrieved in subsequent interactions. Developing for multi-touch interaction means to manage heterogeneous devices, which differ in many hardware and software aspects. From a hardware point of view, it is possible to consider touch-enabled devices according to their form factor. By adopting the original taxonomy proposed by Weiser [1991], we can recognize that modern smartphones are the materialization of *tab-sized* devices. They feature small touch-screens that are both the place where the interface is visualized and also the input device used to operate the system. These devices are personal and portable, can interact with others devices and with the *Cloud* to allow users to store and retrieve personal data. They also embed a wide range of sensors that can be used to *sense* the physical environment, allowing reality-based interactions, location- and context-awareness. Examples are iPhones, Blackberries, Windows Mobile devices and in general all type of modern smartphones. Interaction techniques on mobile devices enable the user to move the content up or down by a touch-drag motion of the finger. For example, zooming in and out of web pages and photos is done by placing two fingers on the screen and spreading them farther apart or bringing them closer together, a gesture known as *pinching*. Other user-centered interactive effects include horizontally sliding sub-selection, the vertically sliding keyboard and bookmarks menu, and widgets that turn around to allow settings to be configured on the other side. "*The next step up in size is the pad, something of a cross between a sheet of paper and current laptop and palmtop computers*", as the same Weiser [1991] reported. *Pads* are a cross between personal computers and smartphones: they offer the same ubiquitous capabilities of a smartphone, but with a large touch screen and a higher computational power. Examples are all kinds of tablets devices, like the iPad or Microsoft Surface Tablet. The last form-factor is *board*: these devices feature medium to large interactive display screens, they are fixed and several people mostly use them in a shared space at the same time. An example of such devices is the DiamondTouch [Dietz and Leigh, 2001], amongst a wide variety of interactive tabletop devices [Kunz and Fjeld, 2010]. DiamondTouch (Figure 2.9) supports small group collaboration by



Figure 2.9: DiamondTouch interactive tabletop.

providing a display interface that allows users to maintain eye contact while interacting with the display simultaneously. It was first created in 2001 as an experimental multi-user interface device. It employs front-projection and uses an array of antennas embedded in the touch surface. Each antenna transmits a unique signal. Each user has a separate receiver, connected to the user capacitively through the user's chair. When a user touches the surface, antennas near the touch point couple an extremely small amount of signal through the user's body and to the receiver. This unique touch technology supports multiple touches by a single user (e.g., two handed touch gestures) and distinguishes between simultaneous inputs from multiple users. *Board* devices often support input from tangible objects, which can be recognized and tracked using electrical- (e.g., Radio Frequency Identification) or computer vision-based techniques (e.g., fiducial markers). Examples are multi-touch horizontal surfaces such as the *reacTable** [Jordà et al., 2007].

2.3.1 Implications for Interaction Design

In ubiquitous scenarios, multi-touch devices having different form factor coexist and communicate: considering the ubiquitous meeting room scenario exposed in Section 1.1, an ecosystem is created by the presence of interactive shared video walls and tabletops and personal tablets or smartphones. In such a context the challenges are technical and conceptual. First of all it is compelling to define a common communication protocol in order to achieve close integration of devices. From a conceptual point of

view, the research question is *"how best to combine devices to achieve a desired type of collaborative user experience"* [Coughlan et al., 2012]. Multi-touch surfaces are integrated with the environment and, therefore, the surrounding space can be *sensed* in order to enable interactions *above*, *behind* and *around* the surface. Microsoft SideSight [Butler et al., 2008] is an interesting project that aims at augmenting touching capabilities of small devices with a proximity sensor. In this way, users can experience multi-touch interactions pinching, sliding and tapping on a paper, tabletop or even the air that's next to the device. The idea of taking advantage of the appropriated surfaces when interacting with small digital devices has been later developed by Harrison [2010]. He envisioned that devices with small (or even without) digital displays, can be made easier to use if we steal surface area from the environment. In his work he surveyed three techniques and related prototypes developed at Carnegie Mellon University's Human-Computer Interaction Institute: *scratch input* [Harrison and Hudson, 2008], with devices detect gesture by listening to acoustics changes a fingernail produces by passing on a surface, *minput*, incorporating optical tracking sensors to capturing motions and device gestures and, *skinput*, providing *"touch-screen-like experience in the body"* by means of a pico-projector.

From a software perspective, protocols are needed to support the recognition of touch input and combine touch points into complex gestures. Different devices run different Operating Systems (OS), like iOS, Android and Windows Mobile. Each OS provides its own protocols and methods to handle touch events: the only cross-platform alternative up to date is the TUIO protocol [Kaltenbrunner et al., 2005], which has been developed to provide an abstraction of underlying input events and therefore promote interoperability between interactive display devices. The TUIO protocol was firstly designed within the reacTIVision project for *"encoding the state of tangible objects and multi-touch events from an interactive table surface"*, as the authors write in the project website ¹¹.

2.4 Tangible Interactive Technologies

With Bricks, Fitzmaurice et al. [1995] introduced the concept of Graspable User Interfaces, later formalized in Tangible User Interfaces (TUIs) by Ishii and Ullmer [1997]. The main idea of TUI lays on physical objects of the real world used as input devices to interact with a computational system. In particular, the Bricks system allows users to direct control virtual objects through physical artifacts, which act as handles for control (Figure 2.10). The innovation is that, while these artifacts belong to the physical world and do not really have digital properties, they can be strictly connected to virtual objects for manipulation or for expressing action (e.g., to set parameters or to initiate a process) and, therefore, they can be considered as new input devices. Moreover, the physical state

¹¹<http://reactivision.sourceforge.net>

by interacting with a touch sensitive whiteboard (SMART technology¹²) using physical elements such as Post-it notes and annotating information by means of electronic pens. The collaborative space was augmented with a computer vision technique, employing a rear-mounted video camera for capturing movement and a front-mounted high-resolution camera for capturing ink.



Figure 2.13: Tangible interface of the AudioPad system.

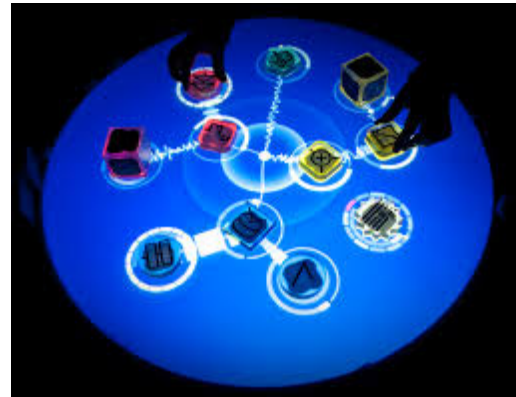


Figure 2.14: reacTable* tangible objects and topological interface.

Systems like AudioPad [Patten et al., 2002] and reacTable* [Kaltenbrunner and Bencina, 2007] are two other examples of tangible interfaces on a multi-touch enabled tabletop. In particular these two systems, which were developed as devices for musical performances, share the same rationale: they use physical tokens having physical and virtual affordances, to direct control digital objects and interact with a tabletop surface. The main differences between the two approaches are in the kind of token they employ, the tracking system and their interactive capabilities. AudioPad uses radio frequency tags and calculate the tags resonance with specially shaped antennas so to obtain very stable 2D position data. reacTable* uses a computer vision approach, employing fiducial markers physically attached to the tokens and an opensource tracking software (reacTIVision) that recognize cartesian and rotational placement of the fiducials on the tabletop surface. Each reacTable* object represents a modular synthesizer component with a dedicated function for the generation, modification or control of sound. One of the most interesting interactive features of the reacTable* is that the resulting sonic topologies are represented on the table surface so to provide a permanent visual feedback. Auras around the physical objects bring information about their behavior, their parameters values and configuration states, while the lines that draw the connections between the objects, convey the real waveforms of the sound flow being produced or modified at each node. Interactions with tangible elements in a multitouch tabletop environment are not only limited to the music world; the same reacTable, for example, has been used for the design and authoring of Role Play Games (RPGs) adventures, using real objects mixed with virtual world. Jordà

¹²<http://smarttech.com/>

et al. [2010], after having developed the reacTable* prototype, are now focusing their research on tabletop and tangible interaction. They are studying how these type of interfaces can favor multi-dimensional and continuous real-time interaction, exploration and multi-user collaboration and also the potential of surface computing in areas such as edutainment, children, elder people and special education. Mazalek et al. [2008] also study the possibility of employ tangible tabletops interfaces for gaming. They developed the TViews Table Role-Playing Game (TTRPG), a digital tabletop role-playing game that runs on the TViews table [Mazalek et al., 2006] allowing to play Massive Multiplayer Online RPGs (MMORPGs) by means of multi-touch inputs and managing tangible real elements.

2.4.1 Implications for Interaction Design

As the development of tangible and physical interaction demonstrates, the focus has moved over the years from the concept of using physical object as input devices to the need of providing a physical embodiment of digital data and embeddedness in real spaces and contexts. TUI are often used in combination with touch-enabled surfaces, creating an environment where the boundaries between the physical and the digital are blurred. This, again, is one of the objective of ubiquitous computing. Nevertheless, as in the case with touchless interaction, it is still not clear how tangible interfaces improve the user experience and what are the real benefits of tangible input/output in interaction. As Hornecker [2012] pointed out—*"Their naturalness and intuitiveness was one of the 'selling points' for TUIs when this interface approach was first introduced. But slowly we are starting to realize how much effort is needed to fulfill this promise, and that a different approach may be needed at times. This is because computer systems by their nature are not like the real world, and because systems need to go beyond real-world behavior to be powerful"*. She, therefore, suggests that researchers should ponder on the difficulties and limitations of tangible interaction in ubiquitous settings. Inquiry the design space in search of a solution imply to generate several prototypes of hybrid systems that can be used to test and evaluate design ideas. This issue primarily motivates this research work, especially in such context where design guidelines have not yet been established and there is a real need to evaluate physical affordances when combined with digital information.

2.5 Libraries, Toolkits and Frameworks for Ubiquitous Interaction

To build an ubiquitous interaction in technologically-enhanced space, novel frameworks and toolkits are needed that support the design and development process [Greenberg, 2007; Marquardt et al., 2011]. By enabling developers and researchers to quickly create

touchless, multi-touch and tangible applications, these tools allow to explore the design space for ubiquitous interaction in order to discover, define and refine design issues and opportunities. Such tools should expose UI functionalities via a well-defined API that would be able to separate basic mechanisms from application specific functionalities.

The state of the art of libraries, toolkits and frameworks is reviewed in this Section according to the interaction techniques that characterize ubiquitous scenarios: touchless, multi-touch, tangible interaction and physical computing. Related surveys on the state of the art of software support to ubiquitous input can be found in the works of:

- Endres et al. [2005], for a survey of 29 software infrastructures and frameworks for the development of UbiComp applications, with a special focus on augmented reality, intelligent environment and distributed mobile systems.
- Figueroa et al. [2005], for a survey of 16 toolkits for virtual reality.
- NUI [2009], the NUI community book on multi-touch hardware and software technologies.
- Klemmer and Landay [2009], where they characterized requirements for physical input by selecting 24 applications and *"categorizing them by four traits: input technology, input form factor, output form factor, and how tangible input and electronic output are coordinated"*.

Before presenting the critical revision of existing technologies, it is essential to highlight characteristics and differences between the concepts of *library*, *toolkit* and *framework*. In the literature they are often used as synonyms, while each one presents a precise shade of meaning.

2.5.1 Library, Toolkit and Framework

In terms of Object-Oriented software reuse, a *library* is a set of implemented independent functions, which can be called by any software application that is using the library. An example is the `java.lang.Math` library in the Java OO language: `Math` is a class that includes several methods for mathematical operations. Therefore, a library is a collection of functionalities, which can be organized into an API, that developers can call in their applications. A *framework* not only provides functionalities but also define a conceptual and software architecture. Although there is no accepted definition for a software framework, in this work the following from Johnson [1997] will be used: *"a framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact"*. The purpose of a framework is to ease the development of applications and improving the quality, reliability and robustness

of new software. The only substantial difference between a software library and a software framework is the concept of *inversion of control*. When a developer is using a library he makes calls to library code and he is in control of the instantiation and invocation of the methods implemented by the library. Therefore the developer needs to know the objects to instantiate and the methods to call to achieve his goal. In a framework the control is inverted: the flow of control for the application is defined by the framework, and there is just a group of predefined white spots that the developer can fill out with his code. The developer implements the objects and methods that are custom to the application and that are instantiated and invoked by the framework. A common way to customize framework behavior is to override framework-implemented features. There is neither a unique definition for the term *toolkit*. The word *kit* suggests modularity and therefore a toolkit is composed by a set of independent libraries. Nevertheless, while these libraries are independent, they are also integrated within the toolkit. This guarantees that the libraries will work well together. In general, the scope of a framework and a toolkit is of a higher level with respect to a library. The software implementation represents the practical manifestation of design ideas expressed by the model of framework or toolkit via its API language. The conceptual level of a framework offers a medium to consider design under a specific point of view and its language influences and shape the way designers approach to a particular problem, opening up to new perspectives and thus promoting creative thinking [Greenberg, 2007]. It is foreseeable that different applications can be developed using frameworks that offer different architectures to address the same problem [Wu et al., 2012].

Having defined the boundaries for the definitions of library, toolkit and framework, the software support for ubiquitous interaction according to the three interaction technologies described in the previous Section 2.2, 2.3 and 2.4 is discussed right after.

2.5.2 Touchless/Remote Interaction

Touchless/Remote interaction with screen displays requires a sensor-based and multimodal approach. At present time, there are no accepted standards, paradigms or design principles for remote interaction with large, pervasive displays. Nevertheless, the interest of both industry and academia in the emergent use of such interfaces is clearly encouraging the growth of communities of human-display interaction researchers [Bellucci et al., 2010]. Table 2.1 lists emerging software support to remote interaction with screen displays.

Microsoft Touchless. Microsoft Touchless¹³ is an open source Office Labs Grassroots library prototype that can create a touchless environment. The Touchless Software Development Kit (SDK) lets developers create touchless-based applications using a

¹³<http://touchless.codeplex.com/>

Hardware	Name	Languages	Features	Type
Webcams	Microsoft Touchless	C#	Objects tracking using color markers	Library
Wiimote, Kinect	GlovePIE	Scripting	Supports basic interactions with Nintendo's Wii Remote and Kinect	Library
OpenNI compliant hardware (e.g., Kinect and Wavi Xtion)	OpenNI	C++, Java	Raw data from RGBD cameras Full body interaction with skeletal and hand recognition and tracking	Framework
Kinect	Kinect SDK	C++, C#	Raw data from RGBD cameras Full body interaction with skeletal recognition and tracking	Library

Table 2.1: Examples of development kit and libraries for toucheless interaction.

webcam and visual tracking fiducial markers. This SDK is free and open-source so that everyone can contribute or adapt it to his own needs. The development environment lets the user configure different markers to be tracked by recognizing the unique color assigned to them. It has been coded in the C++ language and it comes with some demo applications, which helps the developer to understand the logic of the library. The main disadvantage of this library is that it only works with color-coded markers: the usage of the library is therefore limited with respect to real applications, but it is still valid for exploratory purposes, as demonstrated by the prototypes developed by the programmer community¹⁴.

GlovePie. GlovePie¹⁵ stands for Glove Programmable Input Emulator. It was developed to emulate classical computer input hardware. Although it was originally intended for virtual reality data gloves, the library now supports a wide range of input hardware for touch less interaction, including the Wiimote controller and the Kinect. The library features a specific scripting language and a development environment. The environment provides a window where the developer can directly write the code and execute it. Using the scripting language is easy even for non-technical users and therefore is quite straightforward to use a wide range of input devices supported by GlovePIE to interact with different desktop applications. The library also offers a collection of existing scripts, that can be used as starting point to develop more complex interactions. The main limitation of this library is that it can only offers support to the emulation of input devices of the desktop computing paradigm, such as mouse and keyboard. Therefore it does not

¹⁴<http://touchless.codeplex.com/wikipage?title=Community&referringTitle=Home>

¹⁵<http://glovepie.org>

really support the prototyping of ubiquitous interaction systems because it allows only to create hybrid systems that translate touchless input into GUI behaviors.

OpenNI. OpenNI¹⁶ is an open-source software framework that provides an API for developing natural interaction applications using RGBD cameras and audio capture hardware (a microphone array). It has been developed by a non-profit industrial organization led by PrimeSense¹⁷: the company that produced the core sensor of the Microsoft Kinect technology. The same sensor is also featured by the Asus Wavi Xtion¹⁸ platform for natural interaction. It supports C++, C# and Java programming languages and Linux or Windows Operating Systems. The software architecture (Figure 2.15) is conceptually arranged around two main API modules that allow communications with: (a) low-level hardware devices such as RGB camera, IR camera, 3D camera and audio sensors and (b) high-level middlewares for video and audio processing.

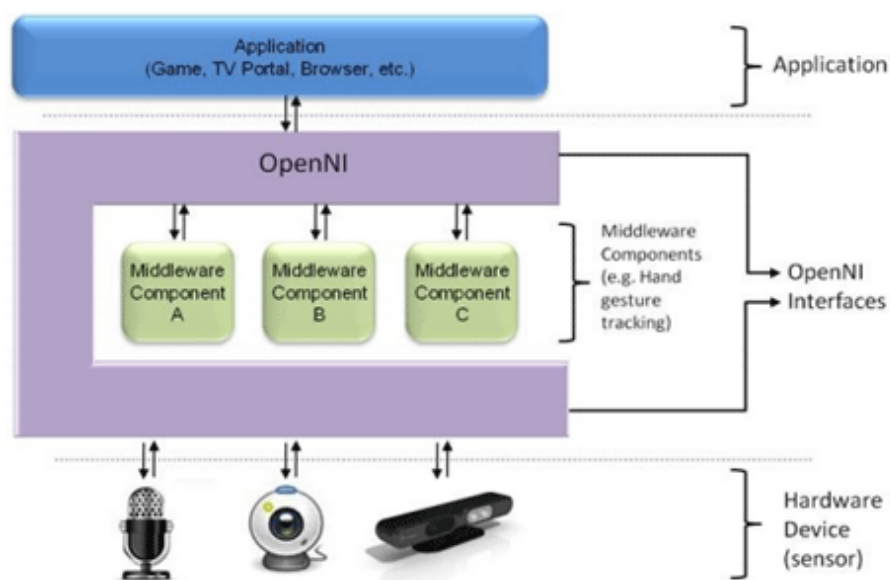


Figure 2.15: OpenNI software architecture.

Therefore, the API provided by OpenNI works at two different levels: at the hardware level it has to be implemented by sensor devices, while, at a processing software level, it has to be implemented by middleware components that supplies high level functionalities for data management. This architecture grants OpenNI portability: sensors and middleware are independent and, therefore, the same middleware software can work on different platforms and receive data with different devices' implementations with no additional

¹⁶<http://openni.org>

¹⁷<http://www.primesense.com>

¹⁸http://event.asus.com/wavi/product/WAVI_Xtion.aspx

coding effort. Algorithms can be written independently of the data source, thanks to the hardware abstraction architecture. Of course, the hardware has to be compliant with the OpenNI specifications. There are two main drawbacks with the framework:

- It supports only a specific class of input devices.
- It can be used only by experienced programmers. Although the API are well documented and there are several examples available on the OpenNI website, the technical expertise required to use the framework is high. For example, the API does not provide an abstraction in terms of data types and if a developer wants to use depth data he has to know how to manage depth map representations in form of bit arrays.

Microsoft Kinect SDK. The Microsoft Kinect SDK¹⁹ is a software library that provides support to the development of touchless systems exploiting the RGBD camera and the microphone array of the Kinect device. The purpose of this library is the same as the OpenNI framework, but being a library it does not provide any software architecture or rationale and, moreover, it is only limited to the Kinect sensor. It works in Windows 7 under the .NET environment and supports the C++, VB and C# languages. The library gives access raw data streams generated by the depth camera, the RGB camera, and the microphone array. Like the OpenNI middleware specification, it also offers the capability to track the skeleton 3D model of one or two people within the Kinect's depth sensor range of detection. Unlike OpenNI, it does not support hand recognition algorithms and therefore it is possible to track only the entire body and recognize body movements but not hand gestures. Lastly, the library provides advanced audio processing features such as acoustic noise suppression and echo cancellation, beam formation to identify the current sound source and the integration with the Windows speech recognition API. The main limitation of Kinect SDK are:

- It supports only one input device: the Microsoft Kinect.
- It does not offer capabilities for hand gestures recognition.
- Like OpenNI, its use requires a high programming expertise. In any case it has to be said that the toolkit offers an extensive support to the developer, with a wide range of reusable components, libraries, tools, and code samples.

2.5.3 Multi-Touch Interaction

A first attempt to the classification of tools for multi-touch development has been made by Kammer et al. [2010]. The main contribution of their work was to present

¹⁹<http://www.microsoft.com/en-us/kinectforwindows>

several criteria to create a taxonomy and evaluate current software support. A review of multi-touch technologies is presented here following the rationale of Kammer et al. [2010] research and focusing on the hardware solution supported, the implementation of the API provided with the SDK (programming languages), the support to tangible objects and implementation of the Tangible User Interfaces Objects (TUIO) protocol. The TUIO protocol, as Kaltenbrunner et al. [2005] stated, *"is an attempt to provide a general and versatile communication interface between tangible tabletop controller interfaces and underlying application layers. It was designed to meet the needs of tabletop interactive multi-touch surfaces, where the user is able to manipulate a set of objects and draw gestures onto the table surface with the finger tips"*. Since its publication for the public domain, many research communities (like for example the NUI group community²⁰) started adopting the protocol in the development of different tangible and touch-based projects. Support to TUIO is important for it is the first attempt to develop a standard protocol for communicating the position, size, and relative velocity of blobs.

Name	Languages	Features	TUI/TUIO	Type
TouchLib	C++	Blob detection and tracking. Fiducial markers recognition. It only runs on Windows.	YES/YES	Library
Community Core Vision (CCV)	C++	Blob detection and tracking. Fiducial markers recognition. It runs on Windows, OSX and Linux.	YES/ YES	Library
libavg	C++, with bindings for Python	XML-based layout language.	NO/ YES	Library
MT4J	Java (Processing)	Abstraction layer. Predefined gestures.	NO/YES	Framework
Multi-Touch Vista	C#	Input management layer.	NO/YES	Library

Table 2.2: Examples of development kit and libraries for multi-touch interaction.

Touchlib. Touchlib²¹ is a C++ software library for developing multi-touch application on Frustrated Total Internal Reflection (FTIR), Diffuse Illumination (DI) and Diffuse Surface Illumination (DSI) horizontal surfaces. For a detailed description of these techniques, which is outside the scope of this work, please refer to the [Schöning et al., 2008].

²⁰<http://nuigroup.com>

²¹<http://nuigroup.com/touchlib/>

Touchlib also supports tangible interaction, by providing basic features for recognizing and tracking physical objects identified by fiducial markers. It essentially captures images from various video sources and processes raw video data by employing some well-known libraries like OpenCV²² to perform luminescent objects (blobs) detection and tracking. It also supports the TUIO protocol used by the reacTIVision library. In this way, it can interface with several other toolkits and programming environments that support this protocol such as Adobe Flash²³, vvvv²⁴ and Processing²⁵. Touchlib can only run on Windows operating system, but an opensource cross-platform version of the library exists, called Community Core Vision (CCV)²⁶. CCV is based on OpenFrameworks²⁷, a software development library which provides platform independent functionality for multi-media development in C++ including support for image capture from video cameras. CCV provides a C++ interface which can be implemented by a class to handle multi-touch input events. It is compatible with the Windows, Mac OS X and Linux platforms but it uses three separate and redundant sets of source code. CCV is one of the most complete and versatile software environments for developing multi-touch interaction; since its last versions it also support tangible interaction via fiducial markers. Nevertheless, it requires high programming skills to be efficiently used and it is difficult to customize. The same occurs with openFrameworks, an open source C++ toolkit for creative coding. It was designed as a general-purpose environment to assist creativity in interacting with digital media. As part of its libraries it also offers support to multi-touch surfaces and tangible interfaces.

libavg. libavg²⁸ is a versatile library for input, processing, and output of audio, video, and image data. It includes support for image capture from video cameras via multiple interfaces for different hardware subsystems. Furthermore, it comes with a large set of image filters including some with GPU support, and contains blob detection and tracking functions. libavg also includes support for scripting using Python. Since it is a library for software development, libavg does not provide ready-to-use functionality for multi-touch systems but only functions to build such systems. A user can use the provided functionality for image capture, image processing and graphical output to develop a multi-touch input system. He needs skills in a programming language or the Python scripting language to solve this task.

²²<http://opencv.willowgarage.com/wiki/>

²³<http://www.adobe.com/products/flashruntimes.html>

²⁴<http://vvvv.org>

²⁵<http://www.processing.org/>

²⁶<http://ccv.nuigroup.com/>

²⁷<http://www.openframeworks.cc>

²⁸<https://www.libavg.de/>

MultiTouch for Java. MultiTouch for Java (MT4j)²⁹ [Laufs et al., 2010] is an open source Java framework, which aims at support different input devices, focusing on enabling multi-touch sensing. It supports the TUIO protocol and therefore it can be used for the detection and tracking of tangibles as well. By relying on Java's virtual machine it features an abstraction layer that allows the development of cross-platform applications.

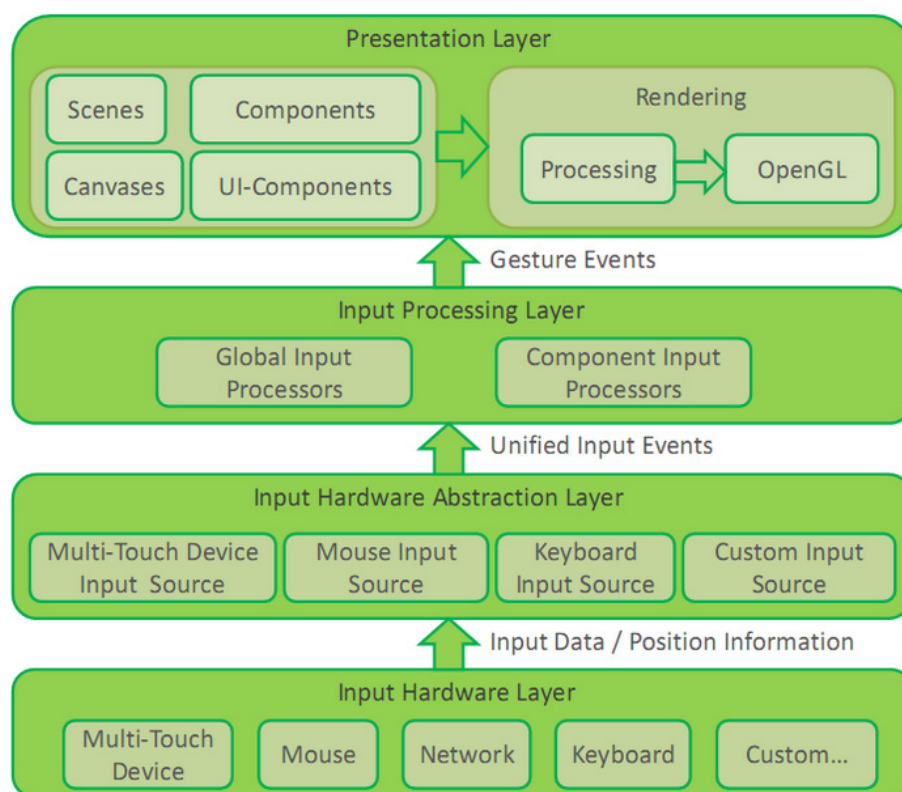


Figure 2.16: MT4j reference architecture.

MT4j defines a reference architecture, shown in Figure 2.16, that is based on the idea of hardware abstraction. On the *Input Hardware* layer, custom drivers for different input devices can be implemented: this feature allows to support, for instance, Windows 7 touch features natively and all the compliant multi-touch hardware and Apple's multi-touch mice and trackpads. The *Input Hardware Abstraction* layer, then, converts the different raw input data into unified input events. In this way it unifies heterogenous input and make it ready to be managed by the *Input Processing* layer. At this stage, gestures listeners are defined that capture the input event and accordingly modify the behavior of target UI components. One peculiar characteristic is the introduction of global gestures: gestures that are congruent throughout the application and may take precedence over component based gestures. Hardware abstraction is a crucial factor to take into account to develop interactive device ecologies. With hardware abstraction, in fact, it is possible

²⁹www.mt4j.org

to design the behavior of the user interface independently of the underlying device that generate the input event.

Multi-Touch Vista. Multi-Touch Vista³⁰ is a .NET input management layer that operates on pre-processed multi-touch input data such as 2D coordinates of a finger contact. It provides an abstraction layer for multiple input data sources and transmits the input data to an application using .NET specific communication mechanisms. It also includes input interpretation logic and a set of user interface controls for use with Windows Presentation Foundation (WPF), a Windows graphical subsystem for rendering user interfaces, and a driver to provide input to Windows 7. Multi-Touch Vista installs its core module as a Windows service which can be configured through a configuration interface by a dedicated application. The library is well designed and flexible. It makes use of multiple technologies (WPF, contracts, unmanaged code, native Windows API, Windows services, etc.), design patterns (e.g., dependency injection), and consists of several separate modules which are barely documented. Thus for programmers with few experience it is hard to understand the internals of the framework. However, it is very easy to use as an application programmer because (a) it integrates into Microsoft Visual Studio, (b) it can be configured using XAML³¹, a declarative markup language that is used for initializing structured values and objects, and (c) it can be controlled using few lines of source code. The main limitation is that the library does not provide a logic to implement multitouch system but it only allows the management of multi-touch input from different sources.

2.5.4 Tangible Interaction

In the last years also a number of toolkits and frameworks (see Table 2.3) to ease the prototype of tangible interaction has been developed. These projects, basically, aim at increasing the capabilities of multi-touch systems by supporting interactions with tangible objects.

Papier-Mâché. The Papier-Mâché³² [Klemmer et al., 2004] toolkit supports interaction with physical objects (*Phobs*) using both barcodes and RFID tags. In Figure 2.17 the GUI of the monitoring application is showing. Papier-Mâché has been the first toolkit to integrate both vision- and the electrical-based approaches in the recognition and tracking of objects. Although it has been principally thought for the interaction with objects, it also allows to capture image streams from videocameras. The main objective of the toolkit is

³⁰<http://multitouchvista.codeplex.com/>

³¹<http://msdn.microsoft.com/en-us/library/ms752059.aspx>

³²<http://hci.stanford.edu/research/papier-mache/>

Name	Languages	Features	TUIO	Type
Papier-Mâché	Java	Optical- and electrical-based recognition.	NO	Toolkit
reacTIVision	C++	Tangible and multi-touch interaction.	YES	Framework
libTISH	C++ with bindings for C#, Java, Python	Cross-platform. Cross-device.	YES	Framework

Table 2.3: Examples of toolkits and frameworks for tangible interaction.

to abstract underlying physical hardware so to shield developer from having to deal with low-level implementations details of cameras or RFID readers. In fact, Papier-Mâché implements a high-level and robust abstraction mechanism that allows to acquire input from different *Phobs* sources and mapping the data into digital behavior via associations. A *Classifier* takes charge of the possible associations, which can be *Nouns* representing content or *Actions*. The vision-based object recognition is implemented using standard computer vision techniques, such as background subtraction and edge detection. This same techniques are the building blocks of vision-based multi-touch systems. Therefore, with very few modifications, Papier-Mâché can also be used to prototype systems that are operated via multi-touch input.

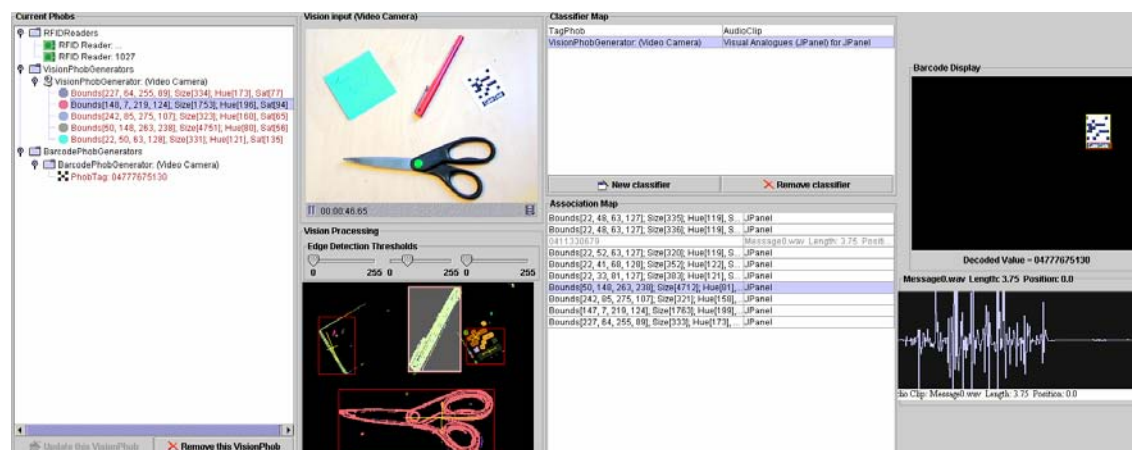


Figure 2.17: The Papier Mache monitoring interface. Source [Klemmer et al., 2004].

reacTIVision. reacTIVision³³ [Kaltenbrunner and Bencina, 2007] is one of the most influential technology that boosted the research and the widespread use of tangible

³³<http://reactivision.sourceforge.net/>

systems. It is a computer-vision opensource framework (see Figure ?? for an operational diagram) the for the development of multi-touch and tangible enabled interactive tabletop, based on the Diffuse Illumination hardware approach. It was designed and developed within the *reactTable** project [Jordà et al., 2007], a tangible and multi-touch round tabletop for musical performances. For being developed especially for this project, the library supports DI solutions and can be partially employed on FTIR platform (in this case only touch input are recognized). *reactTIVision* is a combination of different elements: its main component is a software library that allows quick and reliable tracking of fiducial markers, by processing the raw data received from a video stream. The library employs a specialized computer-vision algorithm for topological markers, firstly introduced by Costanza and Robinson [2003] for the d-touch system Costanza et al. [2003]. Basing on the evaluation of d-touch fiducials, Bencina and Kaltenbrunner [2005] implemented an upgraded version of the fiducial recognizer to be used in the *reactTIVision* library. Their implementation allowed the scalable use of markers in the *reactTable** system, which sizes may vary depending on the number of tangibles required. Together with the main detection and tracking component, the toolkit also provides additional tools like the *TUIOSimulator*, which allows tabletop applications to be tested on a normal desktop computer during development. *reactTIVision* is also the first implementation of the TUIO protocol.

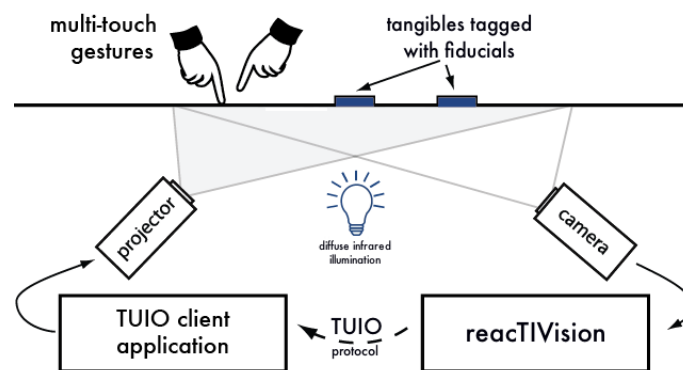


Figure 2.18: *reactTIVision* framework diagram.

libTISCH The libTISCH³⁴ framework [Echtler and Klinker, 2008] supports multiple types of input devices and includes input interpretation and presentation. Figure 2.19 shows the architectural layers, which scheme has been adopted in the framework proposed in this dissertation. libTISCH has a strict separation of layers that are implemented as separate processes connected by sockets using proprietary, compact, and text-based communication protocols. An application can connect to one or in between two layers and process input data by parsing or generating commands using the communication

³⁴<http://tisch.sourceforge.net/>

protocols of libTISCH. The hardware abstraction layer manages and unifies hardware and includes image processing, blob recognition, and tracking. It outputs 2D coordinate data to the transformation layer which applies surface calibration transformations to the data. The interpretation layer detects gestures on the transformed data and signals them to the widget layer. The latter one is designated for integration in a user application and comes with two example widgets which use OpenGL for presentation.

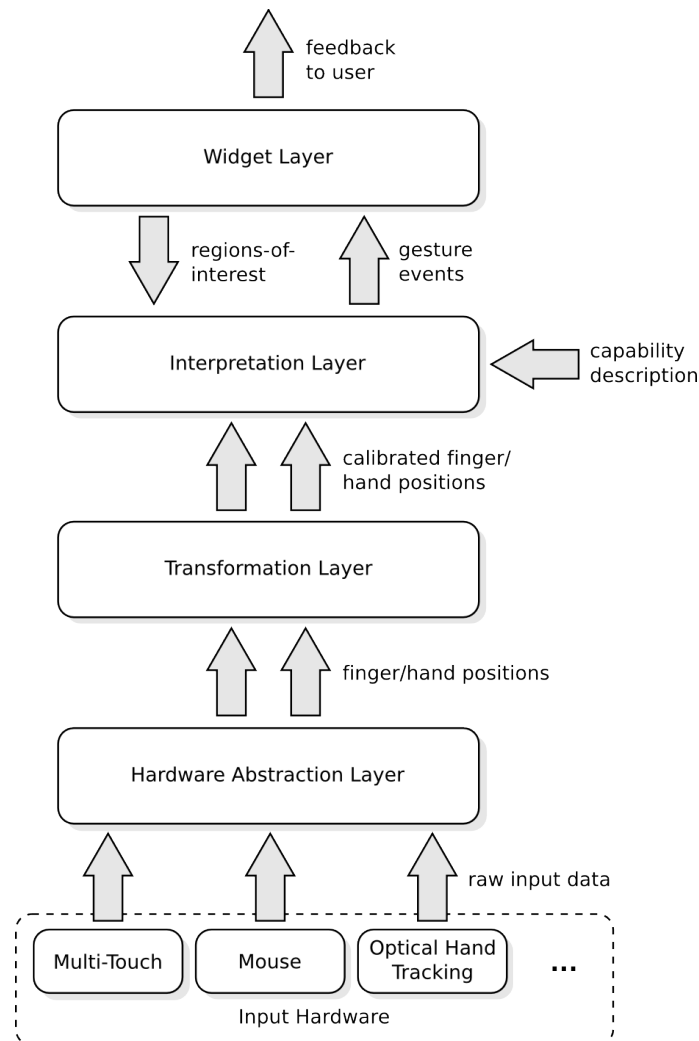


Figure 2.19: The libTISCH architecture.

2.5.5 Physical Computing

As claimed by Gill et al. [2008a], a prototyping toolkits (see Table 2.4) for embedded systems should allow designer to produce a low fidelity prototype within one and two

hours to be accepted by the industry. Therefore, such hardware/software technologies should lower the technological barrier and require low, if none, electronics or programming knowledge on the part of the designer. They should strive for simplicity and ease of use and implement well-know hardware/software design patterns [Greenberg, 2007].

Name	Languages	Features	Type
Phidgets	Several, including C/C++, C#, Cocoa, Delphi, Flash AS3, Flex AS3, Java, LabVIEW, MATLAB	Abstraction of electronic knowledge.	Hardware Toolkit
d.tools	Adobe Flash	Comprehensive environment.	Hardware/Software Toolkit
Arduino	Wiring	Flexible and easy-to-use hardware and software.	Hardware/Software Toolkit

Table 2.4: Examples of toolkits for physical computing.

Phidgets. Phidgets³⁵ [Greenberg and Fitchett, 2001], short for *physical widgets*, is an electronic prototyping platform that has been developed to ease the integration of hardware and software components during the prototyping process. It has been expressly designed for software developers, thus its target user need to have programming expertise. The toolkit physically implements an extensive range of input devices, like buttons, sliders, dials and also sensors such as touch, force and proximity (see Figure 2.20). Phidgets also makes available physical and digital output devices servo motors, relays and small LCD display screen. Each I/O device is mounted on a Printed Circuit Board (PCB) and attached to a central micro controller board which in turn connects to a computer via USB. Phidgets most notable characteristic is the wide variety of sensors that are supported by the platform and the abstraction of electronic knowledge needed to connect components. It also provides interfaces for the most common programming languages and environments such as C/C++, C#, Cocoa, Delphi, Flash AS3, Flex AS3, Java, LabVIEW, MATLAB, Max/MSP, MRS, Python, REALBasic, Visual Basic.NET, Visual Basic 6.0, Visual Basic for Applications, Visual Basic Script, and Visual C/C++/Borland.NET. One of its main drawbacks is in the physical dimension of the hardware: each component is quite large and this makes challenging for designers to use the components in prototypes for small portable devices.

³⁵<http://www.phidgets.com/>

the authors demonstrated that designers did not feel comfortable with the interface of the software authoring tool. In second stance, in order to embed the components and wires, the prototype need to be hollowed, as it happens for Phidgets.

Arduino. To help designers and HCI researches to rapidly give life to physical-digital interaction prototypes, several projects have been created, following the End-User Development (EUD) and Do-It-Yourself (DIY) philosophy. Arduino³⁷ [Mellis et al., 2007] is a clear example: an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It is particularly intended for *"artists, designers, hobbyists and anyone interested in building interactive objects or environments"*³⁸. The toolkit consists of two main elements: an hardware microcontroller board (see Figure 2.22) and a software Integrated Development Environment (IDE).

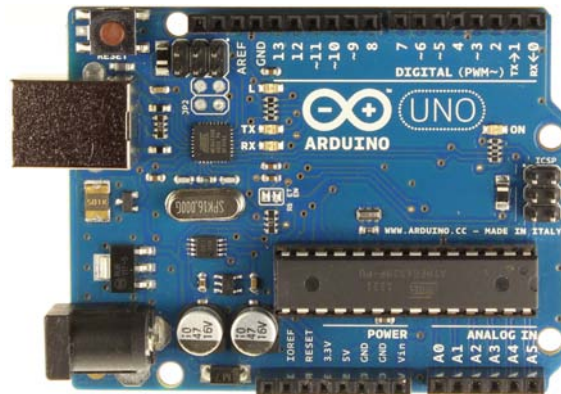


Figure 2.22: Arduino Uno board.

The programming language for Arduino is Wiring³⁹, especially designed to facilitate the creation of sophisticated physical interactive artifacts. Wiring is built on the top of Processing⁴⁰, an open source programming language and environment for people who want to create images, animations, and interactions. Today many users exploit Processing for designing, prototyping, and production. Giving the fact that the Arduino platform is open-source, different design for the microcontroller has been developed, each of them addressing different needs (e.g., physical dimensions or computational power). The micro controller can be connected to a wide range of I/O devices and sensors, receive raw data from them and process such data according to a programmed software. Thanks to its high flexibility and also the support of an active community of programmers, the Arduino platform is very powerful. Nevertheless, it requires knowledge of electronics and also

³⁷<http://www.arduino.cc>

³⁸<http://arduino.cc>

³⁹<http://wiring.cc>

⁴⁰<http://processing.cc>

programming skills to be used. Integration of sensors into a prototype via the Arduino platform, in fact, requires to build the electronic circuit in a PCB, which connects the sensors with the Arduino micro controller, and also to program the software executed on the microcontroller in order to make sense of the data received by the sensors.

2.6 Rapid Prototyping

Prototyping is an extremely important activity for effective design. Prototyping is about rapidly creating an approximation of a design idea so that a designer can promptly get feedbacks about his work. For this reason, prototyping is a central activity that fosters creativity and innovation in design. It allows designers to validate ideas and also to explore a design space, thus promoting serendipity. Through a reflective activity of exploration, by trying things out and learning, one can be able to test and refine hypothesis and also gain useful insights that otherwise it would be impossible to have. Human beings are not good at estimating all possible outcomes, especially in the case of very complex systems. We tend to consider only very few options that are the most likely to occur and most of the time this prediction is biased by our particular background. In design, as well as in other domains, the interaction of all the different variables in play makes it really difficult to predict a result and, consequently, our intuiting on the behavior of a system, or of a user interaction with a system, is often wrong. *"Prototyping is a strategy for efficiently dealing with things that are hard to predict"* [Klemmer, 2012]. Such things can be of two types:

1. Things that you know a priori. In this case the prototype is built specially to reflect and to shed a light on them.
2. Things that designers don't know or consider before and that, actually, come out when interacting with the prototype.

Prototyping allows designers to think in terms of goals and arguing on what they want to achieve with a design idea. As [Buxton, 2007] pointed out, a prototype supports the exploration of the design space throughout the whole design process and helps designers to effectively reflect on the different goals at every stage of the process. Such goals and needs change according to the stage of the process. For example, at the very early phases, designer can be more interested in exploring a wide range of possibilities (e.g., will the product be a desktop service or a mobile service?). When approaching the final product at the end of the design process, they want to narrow down and take into account small variations (e.g., small changes in the layout of the UI). That being said, according to Klemmer [2012], it is possible to define three main characteristics of a prototype in the design process:

- it should not require to be complete,
- it should be easy to modify or change,
- it can be cast aside when no necessary.

A prototype should be generated only to give insights on those aspects we are interested in. For example, if we want to design the user interface for the tactile screen of a digital camera, the prototype does not need to implement all the functionalities of a camera like taking pictures, which requires the presence of photographic elements like lenses and shutters. It should be easy to change because, as said, a prototype is just an artifact that is used as a command ground to discuss idea with different stakeholders. Therefore, as it directly renders design ideas, it is important that it is easy to change or modify with little effort. During the different stages of the design process designers may be interested in achieving different goals, thus a prototype made at early stages could not suit their needs. This is also one of the considerations that should influence the time and effort to build the prototype, depending on the stage of the design process. For example, at the beginning you can be interested in maximizing the amount of learning (insights that you may get) and minimizing the time.

Prototypes are not only used to give insights on system functionalities. In general, according to Houde and Hill [1997], they can express:

- The appearance (*look and feel*), or how does the product look like?
- The functionalities (*implementation*), or how does the product work like?
- The experience (*role*), or what is the user experience interacting with the product?

2.6.1 The Role of Prototyping in HCI

It is important to remember that the output of prototyping activities is not the artifact itself, but the feedbacks that can be obtained from the whole process. The designer builds a prototype to explore an idea, he learns something both building and trying the prototype and, then, he uses what he had learned in the next design. During the years, the role of prototyping has evolved, in the field of HCI, from a tool for the evaluation of a design idea [Jones et al., 2007] to a process that enables designer to reflect on and communicate their experiences during the exploration of a design space [Lim et al., 2008]. This aspect has been also supported by Buxton [2007], who promotes the concept of sketching, along with prototyping, as fundamental activities that drives innovation. It appears clear, therefore, that prototypes are not only viewed as a valid test-bed that can be used to identify and validate requirements, but most of all as design-thinking enablers that allow to discover, define and refine possibilities in a design space. Several different

prototyping techniques have been proposed by HCI researchers and practitioners. One of the first aspects of prototyping that has been taken into account regards *fidelity*, because it is strictly related to cost, both in term of time and resources. The benefit of using low-cost fidelity techniques has been reported by many studies. Low-fidelity prototypes are generally limited function, limited interaction prototyping efforts. They are constructed to depict concepts, design alternatives, and screen layouts, rather than to model the user interaction with a system. In this category falls techniques like: paper-based prototypes for desktop applications, mobile devices and web sites [Snyder, 2003], rapid prototyping for ubiquitous interaction using augmented reality [MacIntyre et al., 2004; Nam and Lee, 2003] and also prototyping for physical interaction [Lee et al., 2004]. Although it has been demonstrated that in many cases a low-fidelity prototype can provide as much insights as a high-level one [Gill et al., 2008b], a low-fidelity approach is not always the best choice, for example in case of in situ tests. That is because the prototype lacks of functionalities or is not similar to the final product [Reichl et al., 2007].

Sketching [Buxton, 2007] is a widely used technique in the design of interactive products. Sketches are particularly useful as a mean to visually externalize design ideas or concepts in the early stages of the design process. Through sketches, for example, it is possible to create storyboards of the interaction between the user and a product. In this way, designers can focus on the tasks the user interface is going to support. Due to the low investment they are an extremely powerful tool to explore possible options at the start and this is directly related to the timeliness, cost and disposability of a sketch. Physical prototypes are less disposable and it takes longer to produce them. Therefore, at the early stage of the design, when there are many different concepts and ideas to explore and, in particular, users tasks and interactions are still uncertain, paper-based sketches are mostly used. Sketch-based prototyping is a popular research area in Interaction Design as demonstrated by the amount of tools to digitally support sketching activities that have been developed over the year. Examples of such tools are SILK [Landay, 1996], DENIM [Lin et al., 2000] and DEMAIS [Bailey et al., 2001].

Participatory design is another popular technique in design, in which the user is an active participant in the design process [Greenbaum and Kyng, 1991; Muller and Kuhn, 1993]. The user is not a professional designer and, thus, prototypes are used to help him in creating and exploring ideas. Examples of prototyping techniques for participatory design are CARD [Muller, 2001] and game-based design [Brandt and Messeter, 2004].

2.6.2 Rapid Prototyping for Ubiquitous Interaction

In ubiquitous systems, the constraints of the classical desktop environment are relaxed and new variables like mobility and physicality are introduced. In Ubiquitous Interaction users are not longer forced to operate the computing system at a desktop but they are free to move in the environment because the terminal evolved into portable, always connected and aware of the context devices (mobility). Physicality of interactive devices also

becomes an important aspect to consider, not only for it defines their physical properties but also because physical affordances influence the way people interact with and rely on those properties. Whereas the Graphical User Interface (GUI) is operated via mouse and keyboard, the Natural User Interface (NUI) makes use of heterogeneous I/O devices (motion sensors, cameras, touch surfaces, etc.) and interaction techniques (touchless, multi-touch, and tangible). In traditional desktop systems, we are accustomed to interact in an environment that inhibits our capabilities. Ubiquitous systems allow more expressive power (by reducing constraints in interaction) and therefore they are expected to provide users with better tools to think, create and communicate. However the design and development of new interactive systems raise both conceptual and practical challenges. In particular, researchers who deal with new technologies need to know different related subjects involving both software and hardware technologies. For example, they should have programming skills, know some basic electronics and also be familiar with hardware drivers, signal processing and communication protocols in order to develop prototypes for tangible and physical interaction. As pointed out by interviews with industrial designers, current support to rapid prototyping of interactive product is not suitable. This is also demonstrated by research efforts on the development of tools that try to ease the rapid prototyping of interaction in systems with different input devices and modalities.

While prototyping tools are common for classical GUIs, as described in Section 1.3, prototyping interaction for ubiquitous systems is still an issue [Wu et al., 2012]. Many frameworks and visual environments have been built in recent years, all of them trying to ease the rapid prototyping of interaction in environment with different input devices and modalities (see Table 2.5).

OpenInterface. OpenInterface⁴¹ [Serrano et al., 2008] aims at providing a visual platform and programming language for multi-modal input processing and interaction. It has been build by a consortium of multiple academic, research, and commercial organizations as a framework to *"handle a rich and extensible set of modalities, enable a focus on new modalities or forms of multimodality, support dynamic selection and combination of modalities to fit the ongoing context of use, and enable iterative user-centered design"*. It is based on a *Pipes and Filters* architecture, which is very common for this kind of visual environments (e.g., vvvv and Squidy [König et al., 2009]). It consists of any number of components (*Filters*) that transform or filter data, before passing it on via connectors (*Pipes*) to other components. The architecture is often used as a simple sequence, but it may also be used for very complex structures. In its last version (last updated in 2010) it supports more than 20 devices and protocols which are also available from an online repository. The components are *"reusable and independent software units with exported and imported input/output interfaces."* and can be developed using a variety of languages such as C, C++, Java, MATLAB, or C#.

⁴¹<http://www.openinterface.org/home/>

Name	Languages	Features	Visual Environment	Type
OpenInterface	C++, Java, MATLAB, .NET	Reusable and independent component.	Pipes and Filters	Framework
Squidy	Java	Automatic data types management.	Pipes and Filter	Framework
vvvv	C#	Extensibility. Support to different interaction modalities.	Dataflow	Software Toolkit
Proximity Toolkit	C#, WPF	Plugin architecture. Distributed data model.	Monitoring Tool	Toolkit
ROSS	Java	Cross-platform. Nested architecture.	No	Framework
VRPN	C	Cross-platform. Device abstraction.	No	Library

Table 2.5: Examples of frameworks and visual environments for ubiquitous interaction.

The visual programming environment provided by OpenInterface are OIDE and SKEMMI. Unfortunately, they are not available anymore for download, therefore it has not been possible to test them directly and we can only rely on their description from Serrano et al. [2008]—“Users will be able to run and quickly modify the multimodal interaction by using the OpenInterface Interaction Development Environment (OIDE) or Sketch Multimodal Interactions (Skemmi) a design-time graphical front-end”.

Squidy. The Squidy Interaction Library⁴² [König et al., 2009] is another input framework and visual environment for the rapid prototyping of multimodal interaction, which is implemented in Java and, therefore, is platform independent. Its architecture is made of three component: the *Squidy Manager*, the *Squidy Designer* and the *Squidy Client*. The Manager follows the rationale of a *Pipes and Filter* architecture, like OpenInterface. It allows to define *Nodes* (this is how *Filters* are called in the Squidy environment) that process the data and *Pipes* that allow the communication of data to other *Nodes*. Conversely from OpenInterface, Squidy provides only a pair of I/O port per node and it includes automatic data type management. The *Squidy Designer* provides the implementation of the visual programming language. Squidy has been thought to be used by designers with low programming expertise; nevertheless, it also addresses the

⁴²<http://hci.uni-konstanz.de/index.php?a=research&b=projects&c=16386645>

need of expert programmers via a semantic zoom over the nodes. The zoom allows the user to directly manage and change the source code of every *Node*, thus giving a high degree of freedom during the prototyping process to both the novice and the expert user. A library of predefined *Nodes* is also available, to drag and drop the desired component directly into the Squidy workspace (see Figure 2.23). Clients are the external applications that are executed on a client device and provide input information to the Manager.

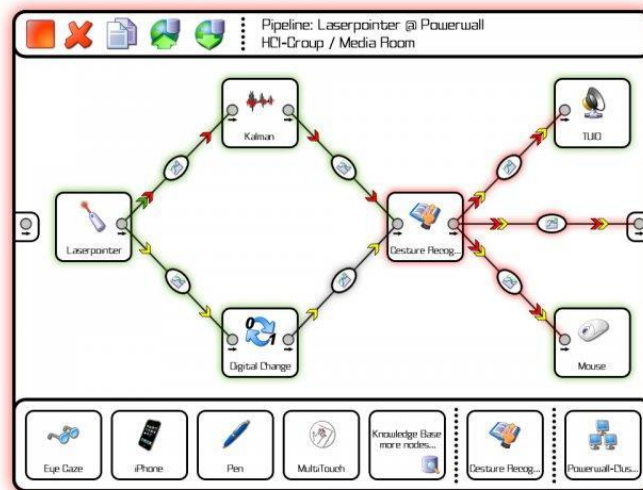


Figure 2.23: Squidy visual programming workspace.

Proximity Toolkit. The Proximity Toolkit [Marquardt et al., 2011] offers developers easy access *proxemic* information from sensors. The term *proxemics* was coined by anthropologist Hall and Hall [1969] to identify the culturally dependent ways in which people use interpersonal distance to understand and mediate their interactions with other people. The theory has been adapted by Marquardt et al. [2011], who used the concept of *zones of engagement*, which characterize how people interpret interpersonal distance (intimate, personal, social, public) to implement different type of interaction between people and the surrounding computational environment. Computational devices react differently according to the distance of the user: ? describe several projects that make use of proxemic information, such as a social surface, a proxemic present and a proxemic media player. The Proximity Toolkit simplifies the exploration of ubiquitous interaction techniques by supplying fine-grained proximity information between people, portable devices, large interactive surfaces, and other non-digital objects in a room-sized environment. The toolkit supports rapid prototyping of proximity-aware systems by supplying developers with the orientation, distance, motion, identity, and location information between entities. It includes various tools, such as a visual monitoring tool, that allows developers to visually observe, record and explore proxemic relationships in 3D space. Its flexible architecture separates sensing hardware from the proxemic data

model derived from these sensors, which means that a variety of sensing technologies can be substituted or combined to derive proxemic information. The APIs are offered via an object-oriented C# .NET development library. It has been designed to be easy to learn and use (1) by taking care of and hiding low-level infrastructure details and (2) by using a conventional object-oriented and event-driven programming pattern. Essentially, the APIs let a developer programmatically access the proxemic data previously observed in the monitoring tool. From the point of view of a rapid prototyping toolkit, the Proximity Toolkit, together with the ROSS API, is the most complete example of holistic approach to ubiquitous interaction.

ROSS. Responsive Objects, Surfaces, and Spaces (ROSS) [Wu et al., 2012] is one of the main research efforts aiming at providing a comprehensive environment for the development of cross-platform ubiquitous interactive systems. It is a toolkit that exposes a software API to allow developers, researchers and designers to create applications in which heterogenous devices communicates through a network. The main rationale of ROSS focuses on a nested architecture in which physical entities are defined in term of spaces, surfaces and objects (see Figure 2.24).

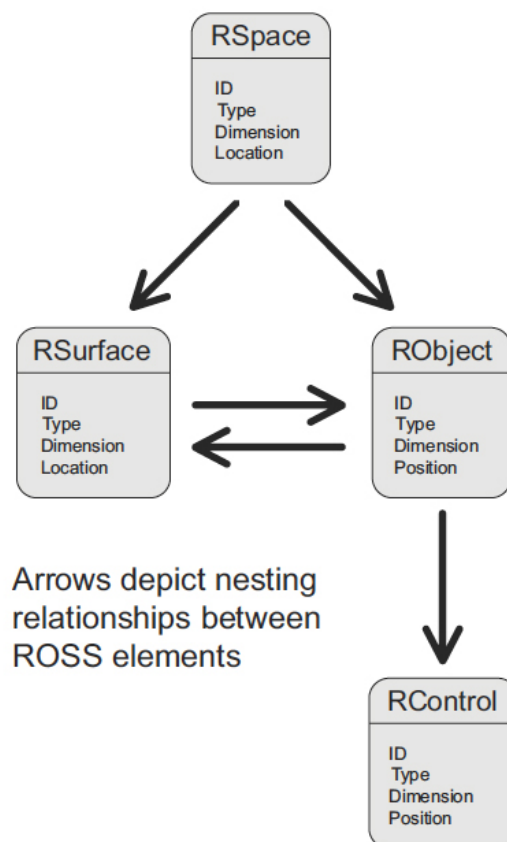


Figure 2.24: Core classes of the ROSS API.

A ROSS-Space (RSpace, e.g., a room) can contains RObjects, physical objects with digital capabilities (e.g., a mobile phone) and RSurfaces, which are interactive surfaces (e.g., the surface of an interactive tabletop). A RObject can nest a RSurface, which in turn can only have nested RObjects. A fourth element, RControl, represents a physical control, such as a button or a knob, and can be attached to RObjects. The nested representation of a ubiquitous environment is defined through an XML configuration file and also the communication between peers is done via the exchange of XML information through the Open Sound Control (OSC)⁴³ protocol. The main benefit of the ROSS infrastructure is its novel perspective. In fact, while *"most APIs are designed to support specific functions and do not interact with other levels in the software architecture"*, ROSS APIs explicitly defines the interactive spaces through relationships between the elements that coexist in that space. In this way it offers a completely new approach to the design of ubiquitous applications. The main limitations are that it only provides a limited abstraction level and that the rationale of the toolkit also limiting the kind of applications that can be developed. From the use case they present [Wu et al., 2012], it seems that the main focus is on applications with screen devices. Moreover, they do not include any mechanism to directly coupling tangible elements with their virtual representations as, for example, proposed by Israel et al. [2011].

VRPN. The Virtual-Reality Peripheral Network (VRPN)⁴⁴ [Taylor et al., 2001] is a software library that has been specifically designed and developed to supply unified access to devices in a VR environment. The authors found that in such systems *"different devices may have radically different interfaces, yet perform essentially the same function; some require specialized connections (PC joysticks) or have drivers only for certain operating systems"*. Therefore, they developed a library that might offer access to a wide range of VR input devices through a general and extensible interface. The approach of VRPN is to provide interfaces to a set of functions, instead of drivers for specific devices. In this way they can implement interfaces and semantics that are consistent among devices that provide a specific function.

vvvv. vvvv⁴⁵ is a *hybrid graphical/textual programming environment* which aim is to ease the prototyping and development of complex media environment and physical interfaces. It uses the *Pipes and Filters* approach and, conversely from Squidy, every Filter node has a number of I/O pins to other nodes. vvvv programs are called Patches and every patch can be used as a node and combined in another Patch to create complex systems. Unlike OpenInterface and Squidy, the GUI is minimal and seems to not respect

⁴³Open Sound Control (OSC) is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology.

⁴⁴<http://www.cs.unc.edu/Research/vrpn/>

⁴⁵<http://vvvv.org/>

established guidelines for GUI design, so that it can be difficult at the beginning to interact with the system and learn how it works. vvvv focuses in visual effects for digital arts, but over the years has been extended to manage input from video sources and image processing algorithm. It can also be used to build multi-touch and tangible interfaces as it supports fiducial marker tracking using the reacTIVision library (see Section 2.5.4). vvvv also offers a wide range of possibilities for extension. For programmers vvvv implement a plugin-based interface which allows nodes to be developed in different languages, like C#, Delphi or C++.

2.7 Summary

The main concepts of interaction in ubiquitous environments, such as technologically-enhanced spaces, have been presented in this chapter. Sections 2.2, 2.3, 2.4 presents an overview of the input technologies that can be used to build device ecologies: in each case example of input devices have been given. The review shows that ubiquitous interaction is characterized by ecologies of networked device that, by communicating one to another make it possible for people to carry out collaborative activities in technologically-enhanced environments. Nevertheless, it makes clear that the heterogeneity of the hardware devices represents a major issue for the development of the ideal scenario in which the interface disappears [Weiser, 1991]. The software solutions presented in the Sections 2.5.2, 2.5.3 and 2.5.4, related to specific input technologies (touchless, multi-touch and tangible) are not suitable for the rapid prototyping of ubiquitous interaction because of the following problems:

- They address one particular device or a class of devices with the same input capabilities.
- They address one specific interaction technique, except for tangible interaction tools that combine tangible objects with multi-touch sensing.
- They target highly skilled programmers.
- They do not take into account interactions between different devices.
- They offer very limited (when provided) abstraction of input devices.

The examination of the software technologies highlights that the development of ubiquitous interaction is scattered among many different tools, programming languages and techniques that the user has to learn and use; at present time only a few software solutions try to manage the intrinsic variety of ubiquitous environments in a unified solution (e.g., the Proximity Toolkit [Marquardt et al., 2011] or the ROSS APIs [Wu

et al., 2011])). Therefore, in the case of tools for the rapid prototyping of ubiquitous interaction (Section 2.6.2), it seems that an important step has been made towards the development of holistic approaches. Nevertheless, it appears clear that two issues are still affecting the smooth development of ubiquitous interactive systems, which are described in details in the next chapter (Chapter 3): the integration of heterogeneous devices and the technical expertise barrier.

3

Open Issues

I'll finish what I started, once.
I'll find my holy grail.

—*The Script for My Requiem*
BLIND GUARDIAN, EPIC METAL BAND

THIS work addresses the concerns raised by Oulasvirta [2008] (see Section 1.3), which are repeated here the sake of convenience and clarity of understanding:

present-day IT infrastructure, “the real ubicomp,” is a massive noncentralized agglomeration of the devices, connectivity and electricity means, applications, services, and interfaces, as well as material objects such as cables and meeting rooms and support surfaces that have emerged almost anarchistically, without a recognized set of guiding principles. This infrastructure is not homogenous or seamless, but fragmented into several techniques that the user has to study and use.

In particular, this dissertation focuses on two open issues regarding the development of ubiquitous systems:

- **Issue 1. Integration of heterogeneous devices.** The lack of a comprehensive framework that provides users with an environment for programming interactions between heterogeneous devices in the same physical space.
- **Issue 2. Lowering the technical expertise for the rapid prototyping of Ubiquitous Interaction.** Prototyping and developing ubiquitous interaction requires to handle low-level details such as information on the position of devices and users in the interactive space and network connections and communications between devices. These tasks require a substantial amount of time and technical expertise, which make the prototyping of ubiquitous interactive systems challenging.

From the literature review, and also from my personal experience as a researcher and developer of interactive appliances, it appears clear that these new challenges are due to the heterogeneous nature of ubiquitous environments, which are characterized by a tremendous variety of input devices, interaction techniques and the fact that these systems are thought to be integrated in a pervasive and collaborative context.

Issue 1. Many frameworks and platforms to ease the design and development of ubiquitous interaction have been built in recent years, as it has been showed in the chapter dedicated to the critical literature review (Chapter 2). Nevertheless, there are very few examples frameworks or toolkits that are based on an holistic approach and integrate all of the different aspects of the ubiquitous paradigm into a comprehensive solution. In fact, even if original vision [Weiser, 1991] conceived a world in with heterogeneous devices (tabs, pads and boards) could communicate seamlessly, all of the actual toolkits focus on a particular aspect of ubiquitous environments, like for example TUIs [Kaltenbrunner and Bencina, 2007; Klemmer et al., 2004] (e.g., Papier-Mâché or reacTIVision) or multi-touch interaction for horizontal surfaces (e.g., MT4J [Laufs et al., 2010]) or physical computing (e.g., d.tools [Hartmann et al., 2005]). Since these approaches give support to a limited range of input devices or interaction techniques, for example tangible input via physical objects with fiducial markers or motion recognition through camera-based software, they do not provide the designer with a unique and integrated environment to seamlessly explore all the possibilities offered by ubiquitous technologies. The only two exceptions in the literature are the ROSS toolkit [Wu et al., 2012] and the Proximity Toolkit [Marquardt et al., 2011]. From the perspective of a unified environment, ROSS still suffer from one drawback: it does not integrate a mechanism to handle the communication between tangible objects and virtual elements. It has been demonstrated by various researchers [Israel et al., 2011; Lifton and Paradiso, 2010] that this is a crucial factor to take into account in an context where physical objects are enriched with digital properties. The possibility to manage virtual representations of tangible objects offers *"the potential to enrich functional and interactive properties of interactive applications"* [Israel et al., 2011]. As Coughlan et al. [2012] pointed out, interacting in environments characterized by a combination of different devices and interaction techniques includes benefits such as *"enhanced learning, problem-solving and creative decision-making — especially where groups of people work together in complex activities"*. To make these benefits happen, *"we need to better understand how people perceive the relationships between multiple heterogeneous devices"*. For these reasons it is important to build better tools that helps in the design, development and evaluation of device ecologies to support collaborative human activities.

Issue 2. Programming environments such as Processing and Wiring, libraries such as CCV and MT4J and software frameworks such as OpenNI or libTISH are intended to facilitate the development of interactive artifacts by providing an API for handling visual and conceptual structures as well as the communication with physical components.

However, although they provide a good level of abstraction, they do not provide general software libraries to communicate with different sensors. You can interface with sensors and get data from sensors, but they only provide raw data that you have to analyze to get some results. Ubiquitous systems make use of sensors and emitters to communicate with the real world. Sensors convert real world inputs into digital data, while emitters are mostly used to provide digital or physical feedback (e.g., a speaker emitting sounds or a blinking Light Emitting Diode—LED). Employing such a variety of hardware devices in a real application can be difficult because their use requires knowledge of underneath physics and many hours of programming work. For example, a digital 3-axis accelerometer is a sensor that gives you acceleration on the three dimensions. Once you get these data, you should interpret them in order to extract some meanings. It is not so straightforward to get the rotation along the y-axis (pitch) from the raw gravity data provided. Furthermore, integrating data from different devices can be cumbersome because any device vendor uses different programming interfaces and communication protocols. This is true also for the same device from different vendors. Imagine that you spent many hours programming the behavior of the accelerometer of a Nintendo Wiimote Controller and want to use the same routines in a new project with the accelerometer of an Apple's iPad. That is almost impossible, due to the incongruous interfaces and protocols used by each sensor. In this case, programming libraries written by expert users can be exploited to interface with these sensors: for example, currently, there is a Processing library for interfacing with the Kinect RGB and Depth cameras and there are also many code samples for getting data from specific sensors. Nevertheless these only represent examples of isolated efforts for providing final users with some libraries for managing sensors data. These attempts do not follow the rationale of a reference architecture or framework and, for this reason, they cannot be structured in a functional API that provides designer with a different perspective to explore the UbiComp design space.

4

Exploration of the design space

I hear and I forgot.
I see and I remember.
I do and I learn.

CONFUCIOUS, CHINESE PHILOSOPHER

THE main objective of this dissertation is to ease the rapid prototyping of Ubiquitous Interaction. Physical objects, augmented with digital computational elements, are more and more present into our living environments, offering the possibility to blur the boundaries between the real and the virtual world. Nevertheless, developing physical/digital interactions is complicated by the heterogeneity of pervasive technologies. In this work, a framework is proposed that can manage the kinds of interactions that take place in ubiquitous environments (e.g., technology-enhanced rooms as discussed in Chapter 2) focusing on the interoperability between heterogeneous devices and data exchange protocols. The framework also considers the space in which users are interacting and the interaction technique in use. Since there is no general acceptance in the literature on the definition and the focus of a framework for interactions in ubiquitous settings [Ni, 2011], the design space is explored in this chapter through the development of real systems. This exploration, together with the analysis of the state of the art (Chapter 2) and interviews with stakeholders, has led to the definition of a set of requirements for the framework.

4.1 Experiences from the Development of Real Systems

In this section are documented the experiences gathered during the last three years designing, developing and deploying interactive systems in the DEI Laboratory¹ at

¹<http://dei.inf.uc3m.es>

Universidad Carlos III de Madrid² supported in part by the national spanish research project Integra³ and Tipex⁴. The report focuses on three prototypes:

1. A touchless system to interact with digital maps projected on video walls [Bellucci et al., 2010].
2. A multi-touch application, running on a tabletop environment, that supports team collaboration for the definition of evacuation routes in case of emergencies [Peralta, 2012].
3. A portable multi-touch system that turn any un-instrumented surface into an interactive surface [Bellucci et al., 2011].

4.1.1 Remote Interactions for Screen Displays with the Wiimote

The Wiimote clearly stimulates the development of touchless post-WIMP (Windows, Icons, Menus and Pointing devices) interactions by exploiting new styles such as gestural input in augmented reality environments. On the basis of its interaction capabilities and low cost, the Wiimote has gained significant attention within the homebrew⁵ software developer and DIY communities, boosting the creation of several projects involving multimodal interaction techniques. These projects are usually shared over the Internet (via www.youtube.com or DIY websites such as www.instructables.com) so others can reproduce and extend them. As MIT professor Eric Klopfer said—*"The advantage of the Wiimote is that it's a human centric device"*. There are many input devices that *"map well onto the computer's interface, but not to the person's"*. By contrast, *"the Wiimote fits the user [...] People know intuitively what to do with it when they pick it up because we use it like devices we are familiar with."* In fact, post-WIMP interfaces are about Minority Report-style interactions⁶, and the Wiimote provides a cheap and effective solution to develop such applications at home or in the laboratory without requiring much more than adequate APIs, small LED-based devices and programming. Therefore, integrating the Wiimote with the surrounding environment can promote the use of pervasive post-WIMP applications even when economic and engineering factors are an issue. In fact, users can develop a solution such as a low-cost multipoint interactive whiteboard with a relatively small budget compared to off-the-shelf systems. Researchers and developers such as Johnny Chung Lee⁷ demonstrated how to develop applications that exploit the Wiimote to perform 3D headtracking, touchless interactions and providing

²<http://uc3m.es>

³(CDTI, Spanish Ministry of Science and Innovation)

⁴TIN2010-19859-C03-01 Spanish Ministry of Science and Innovation

⁵Homebrew is the term used to define software or hardware produced by consumers

⁶For an excerpt from the movie see www.youtube.com/watch?v=NwVBzx0LMNQ

⁷<http://www.johnnylee.net/projects/wii>

haptic feedback. The Wiimote can remotely interact with objects with digital properties on large display surfaces. The GlovePIE library has been employed in this project to map Wiimote inputs to mouse and keyboard outputs. Writing scripts in GlovePIE doesn't require any particular programming experience: Java and C programmers can easily master the language syntax in a short time. In order to explore the feasibility of the Nintendo's Wiimote and GlovePIE for the rapid prototyping of touchless-enabled interfaces, a system has been developed ("*Don't Touch Me*" [Bellucci et al., 2010]) providing the users with the possibility to collaboratively generate and place multimodal annotation on a digital map. The prototype has been designed to be integrated in C4ISR systems to improve user's interaction. C4ISR are the military functions to enable the coordination of operations designated by C4 (command, control, communications and computers), I (military intelligence) and SR (surveillance and reconnaissance). It explicitly addresses the needs of operators of a Command and Control center, who have to examine digital maps on large pervasive displays, manage geo-referenced information and collaboratively develop plans and procedures. The system prototype has been developed on top of Google Maps APIs for cartographic support. The Nintendo Wiimote has been exploited as a primary interaction device and GlovePIE as software library to manage its I/O capabilities. The Wiimote, by means of its accelerometer and infrared tracker, makes it possible to navigate the map via simple hand gestures. Pronate and supinate the wrist while pressing the A button makes the map to displace east or west, while pointing up or down makes the map to displace north or south. You can combine the movement to have the map to displace in all directions. The A button must be pressed all the time in order to avoid unintended movements. Other interactions take place following a point-and-click style, by using Wiimote buttons. *Don't Touch Me* is collaborative in the sense that it supports more than one user (equipped with his own Wiimote) interacting simultaneously. It is possible to identify different roles so to enable/disable users to perform specific actions. In this prototype, two default basic roles are provided: the *editor* and the *viewer*. The *editor* can generate multimedia annotations. He can select geospatial regions by drawing geometrical shapes and assign a color code to each area. Color has the semantic users want to assign to them, in this case: red means emergency, yellow for alarm and green for a normal situation (see Figure 4.1). It is possible to associate interactive events to these sensible areas, by exploiting Wiimote vibratory motor (haptic feedback) and Wiimote speakers (audio feedback). Editors can also record voice annotations (by means of an external audio capturing source, e.g. a microphone) for other users to listen for (through headsets or external speakers). A pop-up pie menu has been used for displaying interactive choices depending on the context. Pie menus (circular contextual menus) are proven to supply a smooth, reliable gestural style interface for users as the circular menu slices are large in size and near the pointer for fast interaction. The *viewer* can only navigate the map and interact with sensible areas (by selecting or simply passing over them with the pointer) for receiving feedbacks. For example, an editor can associate a vibratory feedback to a red sensible area, causing the Wiimote of a viewer to rumble

when passing over such area.

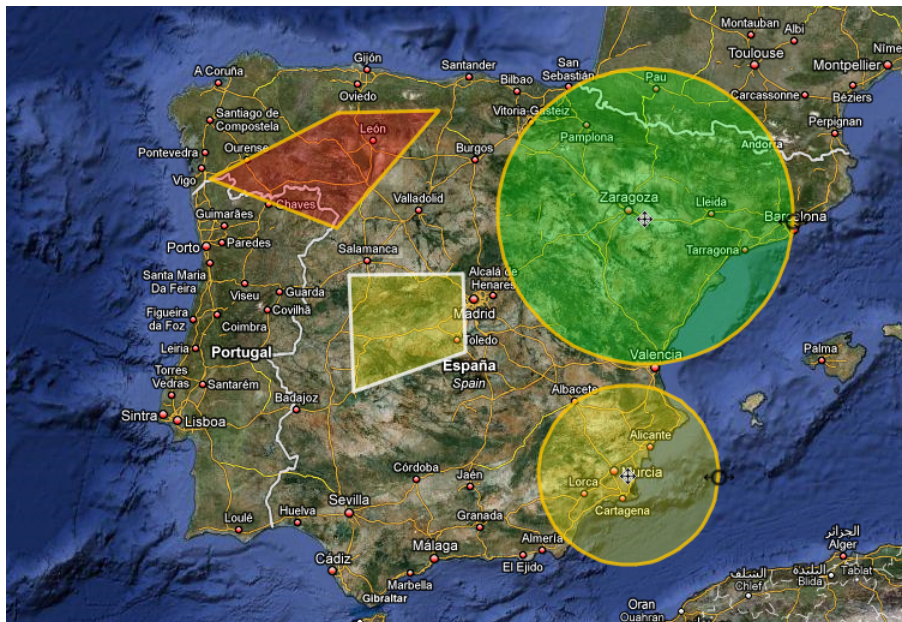


Figure 4.1: Don't Touch Me User Interface.

4.1.1.1 Lessons Learned

From a technical point of view, the development of this application highlighted that, while the GlovePIE works very well for the definition of basic mappings between WIMP actions and Wiimote behavior but it does not offer an high degree of freedom to handle more complex interactions. With the experience of this project, acceleration sensing was intended to be used to built different devices to study how the physical shape of the device affected interaction. Nevertheless this was not possible because the GlovePIE library only recognizes the Wiimote controller as a single, atomic device. There is no abstraction of its components (e.g., accelerometer, gyroscope, speakers, etc.) and therefore, to use accelerometer from other devices, such as Android terminals, the developer is forced to change the programming platform and starting from scratch, with the consequent loss of the code developed. Informal testing sessions with 5 graduate students at University Carlos III de Madrid and 4 attendees at the ACM International Conference on Advance Visual Interfaces (AVI 2010) highlighted that, while hand-free gestural interaction is said to be *natural*, physical controller provide auditory, physical and visual feedback channels that have been demonstrated to be of great help. Overall participants valued the system very positively: for example, they saw the value in using the vibratory feedback to be aware of important areas on the map and then listen to audio annotations left by other users. They liked that they was able to interact with the projected map at a

distance—some compared the system with the Minority Report interface. Users found the interface useful when a group of people needs to work at the same time because the remote interaction does not require the user to be near the surface, thus occluding the vision to others. Some users felt confident with the physical device, although most of them was forgetting to press the A button to start to navigate the map. Others reported on the fact that, while the system demonstrates the potentiality of an interaction which is a step forward the classical keyboard-mouse-display setting, the interface is still bounded to WIMP rules. Therefore, the result is a hybrid environment where novel interaction techniques are applied to an old paradigm. The *Don't Touch Me* use case highlights the need to better understand the role of physical devices in shaping interactions for ubiquitous scenarios and design better interfaces that allow to achieve a desired type of user experience in these environments.

4.1.2 Multi-touch Interactions around a Table with the DiamondTouch

In this project a team of the DEI Laboratory developed a proof-of-concept widget to investigate the use of technology in a real collaborative scenario. The domain was emergency management [McLoughlin, 1985] and, in particular, the widget aimed at supporting collaborative actions that take place during the definition of evacuation plans. We live in a world where disasters (natural or man-made) and accidents are becoming more frequent, causing crises to be solved. Each of these situations must be resolved differently. The proper way to cope and manage crisis and emergency is a challenge for which it is essential to be well prepared and, therefore, only proper planning can achieve a favorable outcome. Depending on the scenario being treated, the way to deal with a crisis situation changes. However, each emergency scenario is characterized by the presence of entities with different roles that must work together to carry out their tasks in a coordinated and efficient way. In our use case we considered the preparation of an evacuation plan (preparedness phase [McLoughlin, 1985]) to face a risk of natural disasters that can affect a large area (such as an urban area) populated by large numbers of people. Since we were dealing with a large area (not a building or a small campus), the number of users involved in the evacuation process increase and a number of different and well-defined roles had to intervene in their basic tasks action. We designed the widget to be used by the following four roles:

- **Civil protection**, responsible of coordinating the various roles in the evacuation process.
- **Local police**, in charge of safeguarding the integrity of citizens, identifying and clearing out those areas where the risk of danger is much greater than the rest.

- **Rescue and fire fighting (firefighter)**, in charge of safeguarding the integrity of citizens and proceeding to the rescue of property or persons in danger of serious harm.
- **Civil guard**, responsible for locating resources needed to transport injured citizens.

According to the groups of actors (roles) involved in the evacuation process, the definition of an evacuation plan envisages cooperation of such users who typically gather around a table and use a map as support for planning. With the map it is possible to identify damaged buildings, display sensitive areas and define escape routes. To digitally support such collaborative activities, the proof-of-concept application exploits ubiquitous devices; in particular the system was implemented through a multi-touch interactive table (the DiamondTouch interactive table [Dietz and Leigh, 2001]), which allows various roles to interact with a digital map. To this end, we have defined a widget, common to all roles, through which they can perform various actions. Through a single widget, visible for all the roles in the system, different functionalities are available (e.g., checking for free beds in a hospital, checking the stability of a given bridge, etc.). Functionalities can be global, like zooming or they can depend on the role that makes use of the widget. The definition of the functionality of the widget is based on the possibility, provided by the DiamondTouch, to recognize which user is interacting with the table, giving it an univocal identification throughout the session. The DiamondTouch hardware, in fact, is based on capacitive coupling [Baxter, 1997] through the human body and a chair connected to the table to close an electronic circuit. In this way it is possible to detect that a touch is coming from the user seated on a particular chair. This feature makes the DiamondTouch a platform to develop tabletop applications for Computer Supported Cooperative Work (CSCW) scenarios that require simultaneous input from different users' role [Esenther et al., 2002]. The development of the widget, thus, provides a new level of management, coordination and collaboration among roles, since they do not rely on a single user responsible for all the tasks but everyone can collaborate in real time to achieve a common goal. As an example, Figure 4.2 shows the functionalities peculiar to the civil protection role such as place an ambulance or mark a building as an hospital.

4.1.2.1 Lessons Learned

This project, in the same way it happened with the *Don't touch me* system, gave me the possibility to experience the limitations of developing an application that targets a specific platform, in an environment where different device coexists and communicates. In this case, the widget has been deployed on the DiamondTouch table, therefore using the specific APIs for the C++ language provided with the platform. As part of an ongoing project at Universidad Carlos III de Madrid, the same widget wanted to be deployed to a vertical surface, so to explore the synergy of horizontal and vertical interactive surface



Figure 4.2: Visualization of the Widget for the civil protection role.

in the same environment. Because of inherent differences of the platform it was not possible to port the widget to the new display device. For this reason an abstraction mechanism has been developed, which is part of the implementation of the framework described in this dissertation. The abstraction allowed to separate the view and the behavior of the widget, using a protocol which is common to any device. Different interfaces have been developed to handle messages from different platforms: these messages, such as TUIO [Kaltenbrunner et al., 2005] or Windows Touch⁸ events, are captured, processed and transformed into common events for the application. Support portability and interoperability between heterogeneous display devices has been the fundamental motivation to develop the framework for Ubiquitous Interaction described in this dissertation.

4.1.3 TESIS: Turn Every Surface Into an Interactive Surface

TESIS, acronym for Turn Every Surface Into an Interactive Surface, is a portable device for converting any surface into an interactive surface. This platform has been developed to overcome the limitations of current interactive tabletops: high monetary cost, lack of

⁸<http://goo.gl/tzY3A>

portability and ad-hoc, flat surface displays. A pico projector and a depth-sensing camera are mounted, on a dedicated stand, above (or in front of) the surface (Figure 4.3).



Figure 4.3: The TESIS system: a pico-projector, a depth-sensing camera and a laptop.

To build the first prototype of the system the Microsoft Kinect sensor has been exploited. The projector is connected to a computing device (e.g., a mobile phone or laptop) to display the user interface. TESIS' depth-sensing algorithm first generates a model of the touch surface and then computes touch inputs depending on the distance from this surface, basing on the algorithm proposed by Wilson [2010] at ITS2010 conference. The touch detection algorithm exploits a naive approach, still achieving good performances for the objective of the project (for a report on the performances of the Kinect depth camera as a touch sensor refer to the research of Dippon and Klinker [2011]). At the first stage a fixed model of the surface is computed over one hundred frames, calculating the depth histogram at each pixel. Then, two threshold are applied to extract touch points as in Wilson [2010]: $d_{max} > d_{x,y} > d_{min}$.

Using two threshold is necessary (1) to eliminate pixels belonging to user's hand (d_{max}) and (2) to balance the noise of the depth sensor when computing the surface model (d_{min}). Contact points are then calculated by subtracting the static model from the actual frame, applying the threshold and processing the resulting image with standard computer vision techniques to extract the final blobs. The result of this process is shown in Figure 4.4.

The goal of this project was to demonstrate that it is possible to use depth-sensing cameras to build novel portable devices that allow digital interaction on every surface

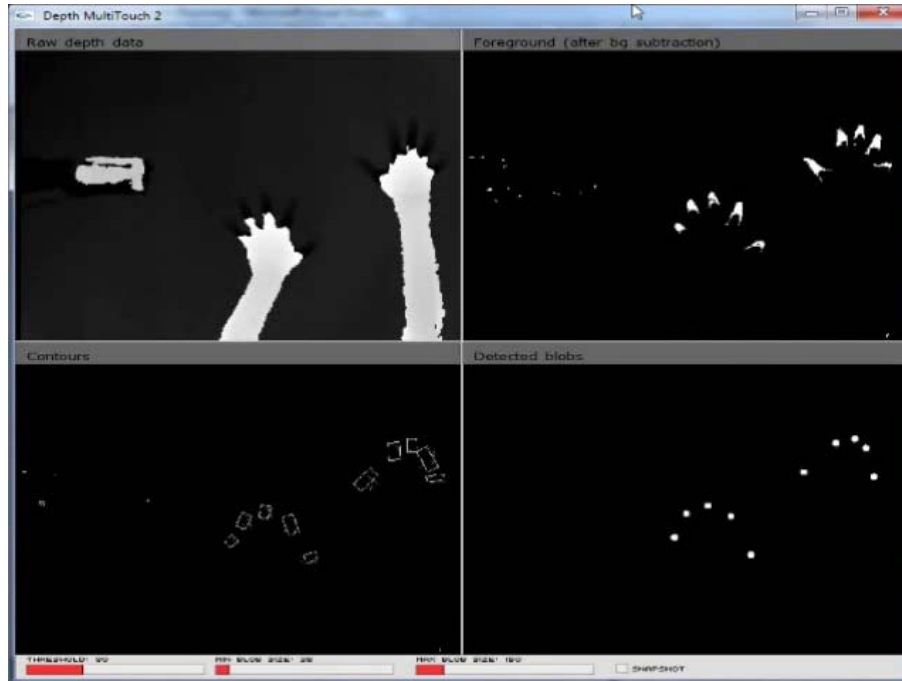


Figure 4.4: Visualization of the recognition algorithm. From left to right, top to down: grayscale depth image, actual image after subtracting the background, contours detection and blob detection.

in the real world. This kind of cheap and ubiquitous devices can therefore support the quick creation of interactive environments where various other devices and interaction techniques coexist. For example, the Kinect sensor provides a RGB camera together with the depth sensor. By exploiting the RGB camera it was possible to integrate mechanisms for tangible interaction, such as the recognition of fiducial markers as provided by the reactIVision framework. The system implements the TUIO protocol, that is: the tracker application capture touch inputs in form of contact points with the surface and the position and shape of tangible objects and sends TUIO message through UDP packets. In this way it is possible to implement client applications that are independent of the platform and of the programming language—they only have to receive the TUIO messages and interpret them in order to enable multi-touch and tangible interaction. Different proof-of-concept applications have been created for the TESIS platform, from a simple application for the manipulation of images to an augmented product counter (see Chapter 6). Moreover, a fixed version of TESIS, with the projector placed on the ceiling and a displaying surface of nearly 60 inches, has been installed on the island of Tiree (Scotland, UK), with the goal of providing a platform to build multi-touch applications for the local rural center and bringing cutting edge technology to the edge of Scotland⁹.

⁹<http://alandix.com/blog/2012/08/20/tiree-touchtable-the-photos/>



Figure 4.5: A fixed version of the TESIS prototype at the rural center in Tiree, Scotland, UK.

4.1.3.1 Lessons Learned

Classical interactive surfaces need to be instrumented, making the setup bulky and fixed (e.g., classic tabletop systems are not portable). As reported above, the motivation behind TESIS development has been to build a solution for interactive surfaces that could be portable. In this sense, current technology demonstrated to be adequate for this goal, even if the precision of the recognition points are lower than classical systems (both electrical- or optical-based). At the beginning, difficulties arose in choosing the development platform because there are many libraries available, all of them covering basic needs for the development with the Kinect. The OpenNI framework was chosen because it is well documented, there is a big community of developer and it follows the rationale of a reference architecture. Moreover, it offers support to different sensors, not only the Microsoft Kinect. This is an important feature thinking on the portability of the code; the scene of available devices for ubiquitous interaction changes rapidly and having a framework that support a wide range of devices is certainly a great benefit for developers who do not need to change their code every time they change hardware. The only requirement is that each vendor makes the device compliant with the framework by implementing its API. The framework, like any other library for the Kinect, has been thought to offer support to skeletal recognition and tracking. In TESIS case, where the depth sensor is placed quite close to a surface (between 80 to 120 cm) there is only low-level support and it was needed to code all the functionalities for touch recognition by interpreting the depth data produced by the sensor. This limits the rapid prototyping, especially if you are not an experienced programmer. On the other hand, for tracking, I was able to use functionalities from the OpenFrameworks libraries which eased the development and highlighted the importance of software tools that provide ready-to-use functionalities for the development of interactive systems. The TUIO protocol pointed out the benefits of interoperability between platforms and devices. Using the protocol I was able to design the system as a TUIO tracker, thus turning it into a development platform for multi-touch, portable applications instead of a simple multi-touch interactive

surface. As reported above, it has been possible to build applications exploiting different environments and programming languages, from C++, to Flash, to Processing. TESIS has been also used in a project developed as part of a Master thesis [Bazoli, 2012], demonstrating how the system supports the rapid prototyping of ubiquitous interactive appliances that combine tangible and digital elements.

4.2 Requirements for a Framework Supporting Ubiquitous Interaction

In his research on the role of software frameworks and toolkit on the development of groupware applications, Greenberg [2007] claimed that:

Interface toolkits in ordinary application areas let average programmers rapidly develop software resembling other standard applications. In contrast, toolkits for novel and perhaps unfamiliar application areas enhance the creativity of these programmers. By removing low-level implementation burdens and supplying appropriate building blocks, toolkits give people a 'language' to think about these new interfaces, which in turn allows them to concentrate on creative designs.

It is interesting for my work to note that Greenberg [2007] identified one of the cause of the failure of development and dissemination of groupware systems to the lack of proper frameworks that support their development. Particularly, he reported that:

- *Most programmers eschew groupware development because it is too hard to do.*
- *Those who do decide to develop groupware place most of their creative efforts into low-level implementation concerns.*
- *Resulting designs are often fairly minimal ones, with little attention paid to necessary design nuances (even ones well known in the CSCW literature) simply because so called 'advanced features' are too hard to implement.*

In this work it is argued that the implementation of ubiquitous interactive systems might suffer from the same problems. As exposed in Section 1.3, the development of a framework for Ubiquitous Interaction has been motivated by the fact that, although there are many software libraries, toolkits and frameworks for post-WIMP human-machine interaction, they do not take into account, in a comprehensive and holistic way, the relationship between the tangible and virtual sides of interactive systems. To this end, once explored the design space through direct participation in the development of ubiquitous

systems, the next step has been to define general requirements for the framework with respect to its various stakeholders [Pressman, 2001, p.255]. A requirement is "[...] something the product must do or a quality that the product must have" [Robertson and Robertson, 2012]. The requirements have been collected as a set of precise statements that describe the issues to be addressed by the framework in order to meet the needs of its potential users. In the next Section the group of possible users is described in details. In any case, the requirements do not represent a solution, but have been used through this research work as guidelines that clarify the issues to be addressed by the proposed solution. In general terms, an input framework has to deal with the management and control of hardware input devices. In the WIMP world the input devices are the mouse and the keyboard, while an ubiquitous setting is characterized by an ecosystem of different devices such as interactive surfaces, cameras, sensors and mobile terminals. The framework is also responsible for the abstraction of input data and its processing. This is particularly important in a scenario where heterogenous devices coexist because it is crucial to handle data fusion and fission [Dumas et al., 2008; Israel et al., 2011]. Therefore, the framework must be able to manage and provide real-time data, using a consistent protocol and data format. The abstraction capability is also pivotal to achieve independence from a particular technology, programming language, driver and API. Figure 4.6 depicts the process used to define the set of requirements for the framework.

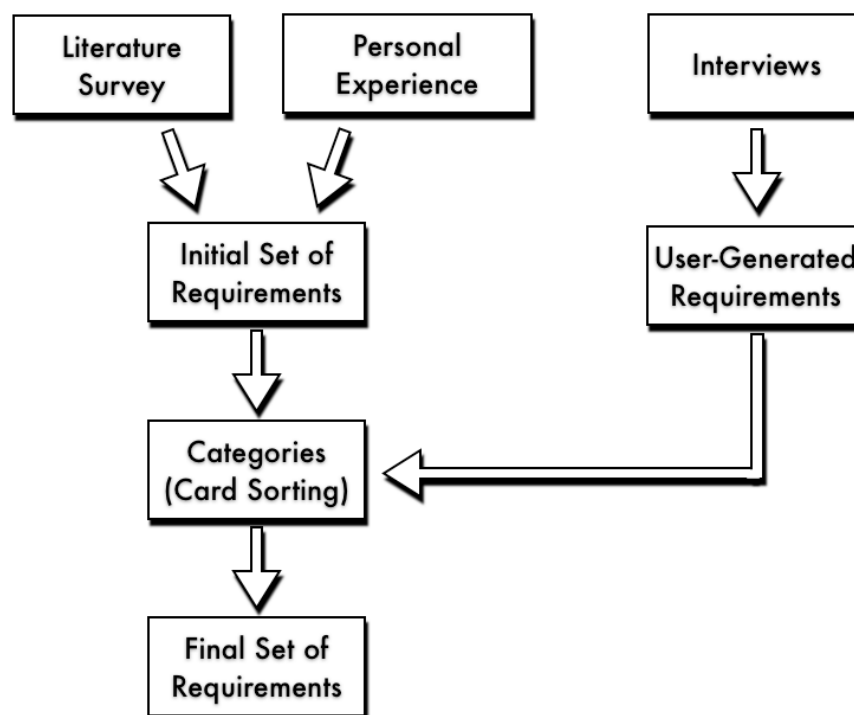


Figure 4.6: The process to define requirements for the framework.

Initial requirements have been extracted from the review of the literature and my personal experience. The state of the art has been analyzed in Chapter 1 and lessons learnt from my personal experience has been presented in the previous Sections of this chapter. User-generated requirements has been also elicited from the experience of developers, interaction designers and researchers familiar with ubiquitous technologies (both from the academia and the industry) through interview sessions. Then, the card sorting technique [Nielsen, 1995] has been exploited to identify categories for the requirements and produce a set of final requirements according to such categories. Card sorting has been described as a quick, inexpensive and reliable categorization method for finding patterns in how users would expect to find content or functionality by sorting cards depicting various concepts into several categories. To follow, in the next section, is provided a description of: (a) the procedure to extract requirements from real users, (b) the card sorting exercise, (c) the categories and, (d) requirements with respect to framework expected functionalities.

4.2.1 Requirements Elicitation: Interviews

Among all the different elicitation techniques with users, such as brainstorming, interviews and focus groups, it was chased to conduct semi-structured interviews [Benyon et al., 2005] to get a specification of the principal requirements for the framework. The rationale behind the choice of an interviewing technique was that it was wanted to extract both high- and low-level requirements, which might cover different domains such as Software Engineering or Interaction Design and which take into account the different perspectives of the users involved, according to their expertise in the field. Therefore, interviews suit best than focus groups, where participants have an homogeneous knowledge—in this case requirements focus only on a particular aspect and a consensus is to be sought. Moreover, at this stage of the design, I needed to be the most open-ended as possible. I wanted to investigate new possibilities for helping people in the development of ubiquitous systems and I started with the assumption, strengthened by the analysis of the state of the art, that current tools do not offer the necessary support. With the interviews I wanted to gain a better understanding of users practices that can guide to the definition and implementation of a framework for ubiquitous interaction. Interviewing is an invaluable technique widely used in HCI and in Software Engineering to elicit requirements by asking users for their direct experiences, especially at early stages of the design process. As Benyon et al. [2005, p.215] pointed out, *"incorporating input from users in the requirements process helps to ensure that the eventual technologies have a good fit with the people, activities and contexts they are designed to support"*. Ubiquitous interaction is human-centered, for it involves the use of technology in every aspect of our daily life. Thus, it is important to include the input from users of the interactive technology in the definition of requirements. This makes it possible to focus on human characteristics and activities during the design. I used the term elicitation because, in this case, the

definition of the requirements set is the result of a process of user-researcher interactions.

4.2.1.1 Goal of the Interview

The first step in setting up an interview process is to decide the goal of the interviews and then to find users that are representative of the target population of the system. To this end, I focused on three questions:

1. Who is going to use the software framework?
2. What kind of interactions they are interested in prototyping?
3. What kind of problems they face off in their practices?

During my Ph.D. research I have been working in a Spanish project aiming at introducing ubiquitous interaction in Command and Control Rooms (project Integra¹⁰, funded by CDTI, Spanish Ministry of Science and Innovation), which are typically operated within the Desktop computing paradigm, and therefore follow the WIMP framework. As I reported in this chapter, I developed several prototypes, including free-hand gestural interfaces and collaborative interfaces for touch-enabled surfaces, that helped me to gain significant knowledge on the benefits and drawbacks of post-WIMP interaction paradigms. I also actively participated in meetings with professionals from the Research and Development (R&D) department of a Spanish company (Amper Sistemas¹¹) that was promoting the paradigm shift from the WIMP world to ubiquitous computation. Thanks to discussions with these professionals, my personal experience, the review of the state of the art and feedbacks from HCI researchers in the Computer Engineering department at Universidad Carlos III de Madrid, I was able to develop a clear understanding of major problems in ubiquitous environments (e.g., lack of a unique infrastructure). For this reason, I did not need to use techniques such as brainstorming to produce preliminary ideas. I instead needed to define the boundaries of a framework for ubiquitous interaction by taking digging deeper into the direct experience of users and know their needs, concerns, practices, preferences and attitudes.

4.2.1.2 Respondents and Users Roles

Table 4.1 summarizes the sample of respondents to the interview according to the three categories of actual users of ubiquitous technologies and potential users of the framework.

¹⁰http://dei.inf.uc3m.es/dei_web/dei_web/index.php?page=projects&id=8&name=Integra

¹¹<http://www.amper.es/section.cfm?id=13&side=139&lang=sp>

In DEI research group there are some Ph.D. and Master students involved in projects which outcome is the design and development of tangible interaction appliances, augmented reality systems and collaborative applications for interactive tabletop surfaces. They represent the user role of *Interaction Developers*: a users that are proficient with programming languages and have a deep knowledge of low-level operations of devices. When using a software framework, the interaction developer needs most of the flexibility and power of the underlying programming language.

During a visiting period at the Cardiff Metropolitan University¹² and the National Center for Product Design and Development Research (PDR)¹³ in Cardiff, Wales, UK, I had the possibility to get in contact with interaction designers with programming skills (e.g., Flash and Wiring) and with an extensive background in physical computing, in particular in the use of Processing and Arduino/Wiring environments. The role of an interaction designer is characterized by casual usage of Interaction Design (IxD) toolkits. They need to produce several prototypes of their ideas rapidly, within an iterative process, and thus an IxD tool should be “easy to use” and easy to learn. They have knowledge of interactive prototyping environments such as Macromedia Flash and can write lines of code if needed but prefer to use visual environments.

Researchers in Interaction Design (IxD) has also be considered as possible users; they are experts of the domain but not necessarily have high technical skills. These users do not want to be aware of the underlying design environment or even the preliminary development efforts. They will use diverse input devices and output devices to frequently interact with their analogue or digital counterparts. Furthermore, excessive learning efforts regarding a specific device can signify its off and if it is the only input device it can lead to an unusable application. Even if they might not have the sufficient programming skill, their insights about goals and barriers in the design of interactive products can be valuable to frame the requirements space for such an architecture. Eventually, in the sample of representative population, a broader set of users has been included, which can be for example interaction designers with no or few experience in programming but that need to develop interactive physical systems. The assumption is that their insights might be useful to highlight possible enhancements of a software framework to be used by non tech-savvy users.

I, therefore, interviewed a total of 10 users, properly selected among the three classes defined above: developers, product designers with programming skills and end-users (e.g., IxD researchers).

At this stage of the design process it was difficult to get access to professional developers and software engineers that were directly working on the development of ubiquitous systems. Therefore, the best approximation was to interview PhD. and MSc. students from the Computer Engineering department. They all have, at least, a four

¹²Cardiff School of Art and Design, <http://cardiff-school-of-art-and-design.org>

¹³<http://pdronline.co.uk>

Role	Interviewee	Profile	Programming expertise	Findings
Interaction Developers	5	PhD. and MsC. students	High	Functional requirements
Interaction Designers	2	Professionals	Medium	Functional and Non functional requirements
IxD Researcher or End-Users	3	University lecturers and professors	Low	Broad view, non functional requirements, design guidelines, future enhancements

Table 4.1: Main characteristic of the respondents population according to target users.

years experience with one programming language (e.g., Java, C++ or PHP) and they all have attended and approved undergraduate courses (or graduate courses in the case of PhD. students) on software engineering, object-oriented programming and design of interactive systems. In any case, their profile still match with the needs of a framework that ease the development of tangible interaction, because during their research they might need to rapidly build prototypes to gain a better understanding of a design space or to evaluate their theories.

4.2.1.3 Procedure

As stated in the previous Section, semi-structured interview style was used. A checklist of topics to cover has been previously prepared (included in Appendix B.2), depending on the profile of the user to be interviewed. This provided a protocol to follow but also the flexibility to change the protocol if I wanted more details about a specific aspect of interest. I first conducted a pilot-testing interview with one users in order to refine questions and make an idea on the potential length of the interview. The interview sessions lasted a mean of 53 minutes (72 minutes maximum and 32 minutes minimum). Before the interview, respondents were briefly informed on the goals of the research (5 minutes) and then they were asked to sign a consent form in order to give me the permission to record the interview and use direct quotations from the transcription of the interview in this dissertation or in other academic publications. Only one of the respondents did not give the permission to use direct quotations. I used the AudioMemos¹⁴ software for iOS to record the audio of the interview and then I manually transcribed it. During the interview notes have been taken. Throughout the interview I sometimes acted as if I was not fully understanding what the person was saying and asked

¹⁴<http://itunes.apple.com/us/app/audio-memos-voice-recorder/id338550388?mt=8>

him to explain more: there is evidence that this approach usually puts the person at ease, encourages him to provide more details, and avoids making assumptions [Jonathan Lazar, 2010]. In general, I conducted the interview starting with broad and high level questions, for example questions about the background of the respondents (e.g., HCI researcher, computer scientist, product designer) and then probing at greater level of details on those aspects of interest for the requirements gathering (e.g., problems encountered developing multimodal applications, particular needs for the prototyping of interactive products, etc.). I asked respondents to specifically recall real situations rather than asking questions about hypothetical scenarios. This approach, especially with more technical profiles, gave me valuable insights on what are the major issues in the development of ubiquitous systems. For the sake of information, I report here on the case of one product designer, who related on his work in real projects where he had to use rapid prototyping techniques to design and develop interactive systems for medical applications (I cannot disclose further details about the projects due to a non-disclosure agreement). His stories helped me in identifying specific circumstances where the framework could positively impact the work practices of industrial designers. In particular, he showed me how he used paper prototyping at the beginning of the design process and how he complained that, as he said "*it provides very low interactivity*". At the time of introducing interactivity, he made extensive use of Flash/Actionscript for the graphical interface and Arduino for the physical interface. He told me that he has a background in product design and that he learned to program interactivity during his Ph.D. by following a learn-by-doing approach. He particularly pointed out that he can use other people's code without any problem, by copying and pasting into his project, but he knows he does not have the knowledge to fully understand how the code works, especially in the case of the implementation of actions that exploit physics laws. With his direct experience in these projects he realized that "*programming equals flexibility*", but it was also aware that only a deep understanding of the code might offer a high level of code reuse in different projects. At the end of each session, a few minutes have been dedicated to give respondents further details about the goals of the research and useful comments that came out from the interview [Kvale, 2007].

4.2.1.4 Analysis

Responses in the transcripts have been breaking down into single thought and ideas, one per line into a text document [Jonathan Lazar, 2010]. Requirements elicited by users have been compared with the initial requirements gathered from the state of the art and the personal developing experience in order to filter out duplicated entries. Overall, requirements from the interviews were resonant with the list of initial requirements and, at the end of the process, 9 new requirements have been added to the initial set of 50. The lists of initial and additional requirements is reported in Appendix B.1 and B.3.

4.2.2 Defining Categories from Requirements

As reported in Section 1.1, interactions in ubiquitous environments depict a socio-technical systems, which are the systematic integration of two sets of design requirements: (a) social components (people) and (b) technical components (hardware devices and software). This work focuses on technical requirements that make it possible to develop an infrastructure to support social interactions in digital augmented spaces. Social components, such as human-to-human communication and interaction mediated by technology, are not addressed by this dissertation. When ubiquitous interaction is examined from a technical point of view, the focus converges on devices that coexist in the same environment thus creating an ecology of interconnected and heterogeneous entities [Coughlan et al., 2012]. Requirements from the state of the art (see the analysis in Chapter 2 for the research works evaluated) and my personal experience have then been extracted following this perspective, resulting in a list of 50 initial requirements, that are reported in Appendix B.1. Each requirement has been backed up with at least one published research contribution. From the analysis of the interviews (Section 4.2.1.4) 9 new requirements have been added, resulting on a set of 59 requirements. The next step was to perform a card sorting exercise [Nielsen, 1995]. The exercise was conducted with 4 experts in Human Computer Interaction and Interaction Design and 2 developers of multimedia interactive systems at Universidad Carlos III de Madrid. Paul [2008] assured that 6 to 12 participants are enough for a card sorting study involving expert users. Participants were provided with an online application for card sorting¹⁵ in which the 59 requirements (from the state of the art and user-generated) were listed, each one with a short description. The list of requirements was randomly ordered by the system for each new session. Participants were asked to group related requirements, to sort them into categories and to provide category headings for these groups. The results from the card sorting sessions was then analyzed using the Average Linkage Cluster Analysis algorithm [Sokal, 1958] provided by the system, deriving a classification in the following 6 different categories:

- **Input/Output Hardware.** Ubiquitous systems rely on different devices to receive input, from touch surfaces to physical objects enhanced with digital capabilities to the whole human body. These same devices are also responsible for the Output, in form of visual representation on screen displays or any form of physical feedback (e.g., haptic). In this category are clustered the different I/O hardware devices the framework needs to support in order to develop interactive spaces.
- **Interaction Modalities.** As in the case of different input devices, the framework must support different interaction modalities in the same scenario. In Human-Computer Interaction, a modality is defined as:

¹⁵The URL of the card sort exercise is: <http://websort.net/s/9D5019/>

- a *sense* through which the human can receive the output of the computer (for instance, touch) and,
- a *sensing device* through which the computer can receive the input from the human (for instance, a depth camera).

In less formal terms, a modality is a path of communication between the human and the computer.

- **Interactive Space.** Ubiquitous interactions take place within a defined space. The framework must be aware of people and devices in such space thus determining position and orientation over time, including mobile devices such as tablets and smartphones and fixed devices such as digital tabletops and wall-sized displays. Awareness of the physical space makes possible to generate virtual representations of the real world and any kind of interaction that requires spatio/temporal data (e.g., proxemic interactions [Marquardt et al., 2011]).
- **Architectural Traits.** The framework is expected to provide a modular, flexible, and easily extensible software design. Similarly to software architecture like Mt4j [Laufs et al., 2010], VRPN [Taylor et al., 2001] or OpenNI, it needs to provide abstraction of hardware devices and support the possibility to connected different middleware for the processing of input data, independently of their implementation. The separation of concerns is therefore compelling, separating how input is acquired from how it is processed. Having different devices requires the need to manage multiple type of data. For example the framework needs to support planar and spatial coordinates as well as motion and acceleration data of objects, supplementary data of multi-touch events such as covered area and pressure, images from video and photo cameras among others. Moreover, this data come from different sources at the same time and therefore mechanisms for the fusion/fission [Dumas et al., 2008] must be implemented.
- **Developing/Coding.** The manifestation of the conceptual framework take the form of a software libraries that helps developers to program an interactive space. The framework, therefore, must expose an API that is concise, easy to use and to learn. The source code needs to be carefully written and to follow clear conventions. It has to make simple things possible with few lines of code and, at the same time, the development of complex things might also be supported within the same rationale. It is particularly important to avoid the Turing tar-pit: that is, an API that allows for flexibility in function but is difficult to learn and use because it offers little or no support for common tasks. As Perlis [1982] wrote in his *Epigramms on programming*:

54. Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy.

Moreover, the framework should also provide tools for users without technical skills to easily develop interactive installations. This can be achieved through visual languages or configuration files.

- **Application/User Interface.** At a last stance, the framework needs to support the development of the User Interface in real applications. Elements in the interface might be the virtual representation of physical objects, thus responding to physical stimuli with changes in the status of the interfaces or can be virtual devices, such for example buttons on a touch-enabled surface, that can provide the input themselves. Due to the presence of different display surfaces in the same environment so the User Interface needs to be distributed.

At the end of the card-sorting exercise, requirements under each category have been further analyzed and grouped into high-level requirements. The process led to the definition of a set of 20 final requirements, listed in Appendix B.5.

4.2.3 Requirements

In this Section the list of main requirements extracted at the end of the process is reported, with an overall description of their scope and how, if applicable, each requirement influenced the development of the framework.

4.2.3.1 Input/Output Hardware

Heterogeneous Input/Output Hardware. As demonstrated by the state of the art and the three applications presented in this chapter, a comprehensive framework for ubiquitous interaction needs an infrastructure that support different I/O technologies that make it possible seamless interaction between the user and the environment. Input devices range from traditional devices from the desktop computing world (e.g., mouse, keyboards, trackpads) to devices peculiar of ubiquitous environments, such as: (a) multi-touch surfaces of different form factors and orientation (e.g., vertical, horizontal or tilted), (b) devices for touch-less or remote input like the Nintendo Wiimote (which features motion sensors like an accelerometer and gyroscope) or camera sensors to capture hand and full body movements (e.g., Microsoft Kinect) and, (c) devices that allows tangible input through the manipulation of physical objects (an example is the Arduino platform). Two of the respondents of the interviews also reported to make extensive use of input from visual markers to enable augmented reality applications. For example, one respondent from Universidad Carlos III de Madrid described the development of the ALF system [Zarraonandia et al., 2012], which aims to improve the communication between participants in a lecture through augmented reality feedbacks. In this system a mobile phone is used by students to send feedback to the lecturer according to the content of

different stages of the presentation. The lecturer can then visualize additional information (e.g., the level of understanding of the explanation) superimposed to each student by wearing augmented reality glasses¹⁶. In this first implementation, each student is identified by the recognition of a visual marker as depicted in Figure 4.7. Lastly, since the landscape

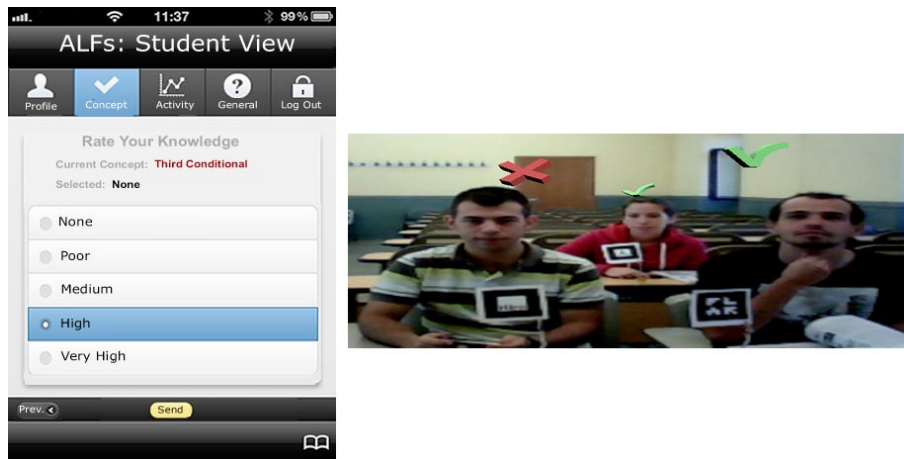


Figure 4.7: Captures of student and lecture views of the ALF system. Source [Zarraonandia et al., 2012]

of available hardware for ubiquitous interaction is constantly evolving, the framework should support novel devices. This issue is addressed by hardware abstraction (as reported in the the *Device Abstraction* requirement under *Architectural Treats* category on page 86) and the rapid configuration of I/O devices (see *Easy Configuration of Input/Output Devices* under *Application/ User Interface* category on page 90)).

4.2.3.2 Interaction modalities

Heterogeneous Interaction Modalities. Ubiquitous systems go beyond the classical WIMP paradigm and empower users to interact with the surrounding computational environment exploiting the higher communication bandwidth offered by novel interaction modalities. As heterogeneous devices coexists, so do different interaction modalities. A framework for ubiquitous interaction, therefore, needs to support different possibilities for the user to interact with the system, from multi-touch, to touchless, to tangible. These three main modalities have been extensively described in the state of the art (Chapter 2). Ubiquitous environments inherently make use of manifold interaction methods, as demonstrated in Section 2.1: the Code Space project present many examples, from the use of personal devices as remote pointers to cross-device techniques (e.g., touch plus air gestures). All of the interviewees reported on their use of input modalities: for

¹⁶The Vuzix Star1200 AR glasses have been used. http://www.vuzix.com/augmented-reality/products_star1200xl.html

instance, one developed distributed applications for co-design activities with children in a multitouch tabletop and multitouch portable devices, another one made use of proximity sensors to develop a sound-based guide for blind people and a last one developed virtual motion sensors for the rapid prototyping of physical artifacts with digital properties. All of the presented methods must be integrated into a single architecture so to offer the possibility of controlling interface widgets using different alternatives for input. To this end, a common data model (*Common I/O Data Model* requirement under *Architectural Treats* category on page 86) needs to be developed that makes it possible to define a common behavior of widgets with respect to different input methods [Echtler and Butz, 2012]. Different modalities can also be used simultaneously and combined together as in multimodal interfaces, where the user can interact, for example, with speech and gestures [Bolt, 1980; Tse et al., 2006]. To enable multimodal interaction, multiple data stream must be managed at the same time and synchronized to implement fusion and fission mechanisms [Dumas et al., 2008]. Proxemic interactions is a recent development in ubiquitous interaction [Marquardt et al., 2011], where spatial relationships of the user's body with respect to other elements in the interactive space (e.g., distance and orientation) are used as input. To support this kind of interactions, an awareness of the physical space is needed, which includes information on the position of users and devices in the space (*Spatial Awareness* requirement in the *Interactive Space* category in the next section).

4.2.3.3 Interactive Space

Spatial Awareness. An ubiquitous system must be aware of people and devices in the room. Spatial awareness is the ability to determine position and orientation over time – including mobile devices such as tablets and smartphones and fixed devices such as digital tabletops and wall-sized displays. The position of fixed devices within the room as well the layout of the room should be specified through a configuration file, which can be generated through a dedicated graphical user interface. The location of mobile devices can be tracked using camera sensors, such as the Microsoft Kinect ,while orientation data can be captured from the gyroscopes already built into many mobile devices. This data can be transmitted back to the system and integrated with the fixed position data to support spatial interactions. For example, when a user attempts to use a *pointing gesture* [Kray et al., 2010] to send an item from a mobile device to a tabletop, the system would use spatial information to select the destination device that the user intended. Spatial tracking allows a system to track the position of people, devices, and other items in the interactive space. Coupled with a model of the interactive space, this information can be used to support a variety of proxemic interactions [Marquardt et al., 2011]. Spatial awareness is also required to enable a matching between the real and the virtual world, for example for gaming or interactive simulations [Izadi et al., 2011].

Multi-Surface Environment. The framework needs to take into account that multiple interactive surfaces of different dimensions and orientations coexist in the same environment. Knowing the spatial position of each surface in the interactive space make it possible to implement interaction between surfaces, for example to transfer content from one surface to another via gestural input. Moreover, it is necessary to determine relative position of objects in each surface, for instance in the case of tangible interaction on a tabletop [Jordà et al., 2007]). In this way it is possible to exploit objects in one surface as input for another surface, as happens in Augmented Surfaces [Rekimoto and Saitoh, 1999] where a physical object, placed on an horizontal surface, representing a virtual camera is used to change the 3D view on a vertical screen.

4.2.3.4 Architectural Treats

Distributed Architecture. In order to work in different interconnected devices, the framework need to implement a distributed architecture. The framework aims at relieving users from the burden of managing low-level details, as in this case, the communications between heterogeneous entities. Therefore, it must provide an abstraction layer for connections and communications that presents a common external interface for all the device. Users should only be aware of the content of the messages and not the way they are sent or received, neither how connections are established and managed over different communication channels (e.g., TCP, UDP or Bluetooth). Devices discovery should be automatically handled by the architecture, again with the objective to ease the programming of the ubiquitous system by making low-level communication details transparent to the user. Communication channels must be full-duplex: they are used to interchange messages to handle to network configuration between devices in the ecology but also to carry feedback information from one device to another. For example, the *Don't Touch Me* presents a scenario in which an input device is used to provide haptic feedback through a vibro-motor when a cursor passes over a sensible area of a display device. Having a full-duplex connection between devices makes it possible to implement this kind of events, where an action on a device can activate feedback on another one.

Common Communication Protocol. A comprehensive framework needs to establish a common protocol that supports communication of all required information between the hardware layer and interaction layer of a ubiquitous interactive system. This provides interoperability between different devices. The protocol must address the need of ubiquitous environments: it has to be flexible, extensible, it must support data generated by all the hardware devices in the *Input/Output Hardware* category and it has to facilitate the implementation of different interaction modalities (see *Interaction Modalities* category on page 84). TUIO [Kaltenbrunner and Bencina, 2007] is an example of such a protocol, developed for multi-touch and tangible interaction. Six respondents to the interviews

identified the need of a communication protocol as a fundamental aspect to take into account for a framework to ease the prototyping of ubiquitous interaction.

Expandible/Extensible Architecture. The architecture must follow the design principle of separation of concerns. It needs to be separated into distinct layers, such that each layer addresses a separate concern. The value of separation of concerns is simplifying development and maintenance of computer programs. When concerns are well separated, individual sections can be developed and updated independently. This enable seamless extensibility of the framework. Two examples of this approach in the case of ubiquitous interaction are Mt4j [Laufs et al., 2010] (see Figure 4.8) and the TUI-VR framework [Israel et al., 2011] (see Figure 4.9). The two frameworks clearly presents separation of concerns, having a layer that handles physical connections with devices by means of dedicated drivers, a layer for the abstraction of physical devices, a layer where the input is processed (*Input Processing Layer* in Mt4j and *TUI-object abstraction layer* in TUI-VR) and an presentation layer (*TUI Application* in TUI-VR). An ubiquitous environment is made of *loosely-coupled* components [Krafzig et al., 2005]—that is, the devices that compose the ecosystem have, or makes use of, little or no knowledge of the definitions of other separate components. The interface between loosely-coupled independent devices should not include any component-specific behavior or state. It can include the data model of the information exchanged between the components, and the role played by each component. The data model is the only invariant (if any) in a loosely coupled system and should be primary, because it allows to implement device abstraction (*Device Abstraction* requirement in this section) and to support third-party middlewares (*Agnosticism of Legacy Systems/Middleware* requirement in this section). An event-driven architecture (EDA, [Philip, 1998]) is a common way to implement data-oriented concepts. In an EDA, loosely coupled components in a well distributed layered architecture process their data in an asynchronous way. By installing conditional triggers, events are generated to signal certain state changes. All components associated with a certain trigger are notified by routing events through specific gateways based on their destination addresses. At the destination the asynchronous events are stored in a FIFO-queue and then processed.

Device Abstraction. Several works highlights the need for hardware abstraction and high-level interfaces to support the development of ubiquitous interactive systems [Israel et al., 2011; Laufs et al., 2010; Taylor et al., 2001]. Device abstraction is the first step toward the integration of heterogeneous devices. By providing an abstraction layer, the different raw input data are converted into unified input events, following a common data model. In this way the input will be handled by the other layers in the architecture independently from the device that generated it. Abstraction also allows seamless integration of new devices by providing abstract super classes of all the input sources to be extended by adding the functionality specific to the new type of input. The

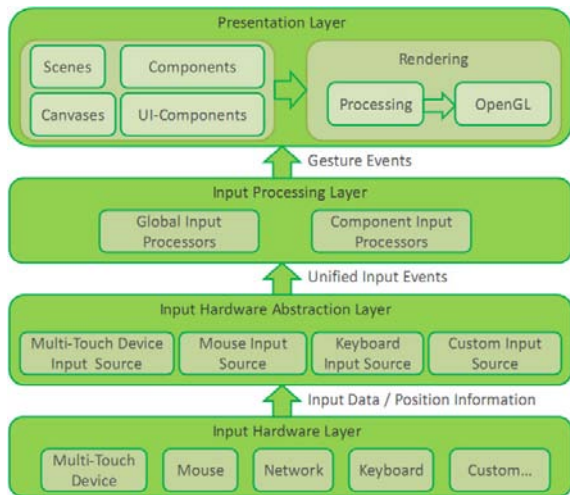


Figure 4.8: MT4j reference architecture.

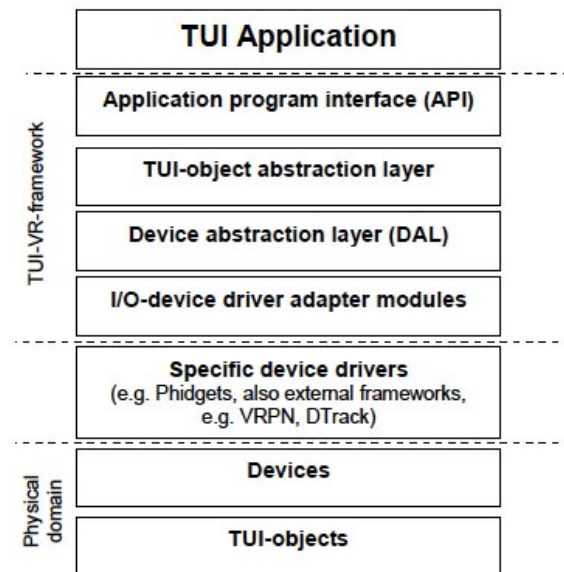


Figure 4.9: TUI-VR reference architecture.

importance of device abstraction has been highlighted during the development of the *Don't Touch Me* system. As exposed in Section 4.1.1.1, changing the input device meant to rewrite the implementation from scratch, even if the same sensor (the accelerometer) was used. One of the interviewees (a developer at the Electrical Engineering department at Universidad Carlos III de Madrid with an MsC in Electrical Engineering and several years of experience developing embedded systems) reported on the same problem. He was developing applications for augmented reality glasses and he complained that when he needed to switch from the Arduino platform to IOIO¹⁷, in order to support Android devices, he needed to completely change the implementation.

Different Data Streams. The architecture needs to manage data streams produced by the different devices. Fusion techniques handle the synchronization of coreferenced input events in multimodal systems. This consists of collecting and integrating data from multiple complementary input streams into an accurate interpretation of the user's intention. A simple example is *Put that there* [Bolt, 1980] in which the vocal command *put* must be linked to two pointing events in the correct order to achieve the desired representation. Fusion is one of the core properties of interaction in virtual environments, as all input events must be interpreted with reference to the location of the user, the interaction device, or both. Fission techniques handle the selection of appropriate output channels for feedback and the construction and synchronization of output into a coherent message for the user. Fission techniques are essential to ubiquitous applications that

¹⁷<https://github.com/ytai/ioio/wiki>

output feedback to users via many different modalities, such as environments that employ ambient media to display selected information to a user or devices such as the Nintendo Wiimote with haptic feedback (vibro-motor). Data Fusion/Fission can be handled by implementing timestamps in each I/O message as suggested by Taylor et al. [2001].

Agnosticism of Legacy Middleware. Middleware provides services to software applications beyond those available from the framework. Two of the interviewees explicitly reported that they found crucial to allow third-party developers to implement their solutions for input handling to be included as pluggable elements in the system's architecture without changing the infrastructure. Middleware receives data (according to the common I/O data model) from the abstraction layer and computes it independently of the hardware. For example, a middleware for 3D hand tracking will receive depth data in the same format from any depth camera supported by the framework. This will increase the modularity of the system.

4.2.3.5 Developing/Coding

Common Programming Language. The framework must use the same programming language for all its components and layers of the architecture in order to lower the technical knowledge needed to program in a heterogeneous environment, foster the integration of different devices and allow the definition of a concise and unique API.

Familiar Development Platform. APIs should *be embedded within a familiar platform and language in common use so that people can leverage their existing knowledge and skills* [Greenberg, 2007]. The framework implementation has been developed in Java, using the Processing platform, which is widespread adopted open source tool for creative programming.

Low Viscosity of the Code. The infrastructure should facilitate iteration and experimentation at both design-time and run-time— that is, to have low viscosity [Wilde, 1996]. At design-time, it is important to be able to test alternative solutions quickly for rapid prototyping. At run-time, it is important to be able to accommodate unexpected situations so that users are not hindered by the capabilities of the environment. Low viscosity is characterized by flexibility, expressive leverage and expressive match. The framework should make simple things achievable in a few lines of code, but also complex things possible by adding design-preserving code to a system.

Hide Low-Level Coding Details. Compiling low-level information, such as the position and information of a device into meaningful high-level information — for example,

out of several possibilities, which device is a user trying to interact with — is a complicated task. These tasks require a significant amount of mathematical knowledge and programming time. To ease the development and support the rapid prototyping of ubiquitous systems, the framework must hide low-level coding details by provide high-level functionalities through its API. This includes also the way the framework manages connections with the underlying hardware. One of the interviewees reported that:

there are various interfaces that can handle communication with hardware, such as Arduino, that have standard libraries of doing this and thus helping the procedure significantly. If not that lucky though and need to implement the solution around a custom controller then it is quite time consuming to get accustomed of programming the controller according to your needs. In solutions where simple systems involve, like keyboard or mouse controllers, then the difficulty gets substantially decreased

Greenberg [2007] suggests to *"encapsulate successful design concepts known by the research community into programmable objects so they can be included with little implementation effort."*

Programming Alternatives. An API is the main manifestation of the conceptual framework and developers will use its specification to code the application. For this reason it should be concise, easy to learn and provide a full and clear documentation, including code snippets and examples. However, alternatives should be provided, such as programming via hard coding, using a configuration file — for instance to define components of an input device — and support user scripting. Visual programming tools are recommended, to support users with low programming knowledge [König et al., 2009].

4.2.3.6 Application/User Interface

Real (e.g., haptic)/Digital(e.g., visual) User Feedback. The framework must provide different feedback channels, taking into account that in an ubiquitous environment the real and virtual world are intertwined. Bidirectional (duplex) interaction between the physical domain and the digital model is a key factor in ubiquitous interaction, especially when tangible user interfaces are employed. Where other framework concepts often require collection of relevant data from different sources and transmission of feedback data to various actuators, ubiquitous frameworks should encapsulate input and output in a single software object in order to make it easy for the developer to implement a physical/digital behavior.

Cross-Device UI. The user interface needs to be distributed in different devices that form the ecology in a way to achieve seamless user experience across devices. A separate language would be introduced to describe UI components; this language would then be interpreted during run-time by the app taking into consideration what the device is capable of and generating the required code. Typically a language of that kind is a XML dialect, but there's also a possibility to use for example JSON¹⁸. The whole purpose of it would be to hide the specific implementations from the developer, so he could concentrate on the actual UI building.

Storage and Replay of Interactive Sessions. The framework should therefore provide a log file mechanism, by which all messages in the session can be stored to file, and then the session replayed or analyzed. Researchers might be interested in having a log of the interactive session to study different aspects of social interactions in ubiquitous settings [Coughlan et al., 2012] such as:

- Record user motion during human-factors studies.
- Store interactions between collaborating users to enable comparisons between different sharing strategies.
- Capture a series of user motions and gestures to enable debugging of new interaction techniques.

Easy Configuration of Input/Output Devices. The framework should provide methods to easily configure devices that form the ecology. A graphical user interface can generate a configuration file with all the necessary information on the devices, hiding low-level details to the user.

UI Widgets. Widgets must expose a common behavior to different input methods. The application layer of the framework should provide predefined widgets as build blocks for the user interface; they should be easy to customize or extend in order to implement complex behavior. The users should also be able to implement their own widgets from scratch.

4.3 Summary

In this chapter, the design space of ubiquitous interactive systems has been explored and lessons learnt from the development of real applications have been reported. With these

¹⁸<http://json.org>

experiences in mind and also grounding on the analysis of the state of the art, a set of requirements has been defined for the development of a framework for ubiquitous systems, which takes into account interaction between interconnected devices. The requirements have been contrasted with the experience and needs of the possible stakeholders of the framework and the original set of requirements have been refined according the analysis of the interview with 10 participants, employing user-centered design techniques. The final set of requirements reveals that a framework for ubiquitous interaction needs to support a wide range of I/O hardware devices, which are also rapidly changing over the time, as well as different interaction modalities. Support to heterogeneous devices implies device abstraction mechanisms that allow to receive input from different sensors and send output to actuators independently of the actual implementation. Device abstraction also allows to define interaction modalities that are agnostic with respect of the underlying input technology. The physical space is augmented with computational capabilities and entities in the interactive space are expected to respond to spatial stimuli, thus they need to be aware of the surrounding environment. Moreover, computation is not yet confined in one single terminal, but it is distributed among the devices that constitute the ecology. The definition and implementation of a distributed architecture is therefore mandatory. The architecture is expected to feature a common communication protocol to guarantee device interoperability and it needs to manage different data streams from all devices' sensors. In order to ease the programming of ubiquitous systems, the framework needs to expose a functional API that abstract from low-level details. Moreover, programming alternatives such as configuration files or visual tools should be provided as well. From the point of view of the application, the framework should provide mechanisms for the easy configuration of input device and linking the behavior of devices to digital elements. Predefined widgets and behaviors should be provided as well as the possibility to seamlessly customize or extend existing functionalities. Different feedback mechanisms need to be supported: ubiquitous environments create a bridge between the real and the virtual world, thus bidirectional communications between physical and digital entities are necessary that allow to generate both real (e.g., haptic) and virtual (e.g., visual) output. Lastly, the framework should also provide storing facilities, because researchers might be interested in having a log of the interactive session to study different aspects of social interactions in ubiquitous settings.

5

Design and Implentation of the Framework

Tat Tvam Asi.

— *Chandogya Upanishad 6.8.7*

IN this chapter details about the architecture of the framework, its design and implementation are given. The framework builds on the prior work of *VRPN* (see description on Section 2.6.2), bridging the connection between the virtual and real world by offering abstraction of input devices and a homogeneous data type hierarchy to define the information exchanged by different devices. The same approach is also shared with the *Proximity Toolkit* (Section 2.6.2). The *Squidy Framework*, *OpenInterface* and *vvvv* (all of them described in Section 2.6.2) highlight the need to provide support to different input devices, nevertheless they are based on implementations tailored on specific devices, while the framework proposed in this dissertation adopts an abstraction of atomic components, which also allows the definition of custom devices. The framework shares with the ROSS API (Section 2.6.2) the configuration of the environment through XML files and the use of OSC protocol to exchange data between different modules. In this stage of the framework, spatial integrity of objects (in other words, the spatial interconnection among objects, such as their absolute position in the environment or relative position with respect other objects) it is only supported to a very limited extent, while it is a main characteristic of Proximity Toolkit and ROSS. Full support to the spatial dimension will be implemented in future works. The framework also builds on the reference architecture of MT4j (Section 2.5.3) for the idea of unified input events and OpenNI (Section 2.5.2) for the capability of managing data at a middleware level, independently of the underlying implementation of input sensors . In the following sections of this chapter, the underlying interaction model involving physical and digital elements is presented together with the architecture of the framework. For each layer of the architecture, the contribution of the framework with respect to the state of the art is highlighted and details of the current

implementation are provided. Finally, a table is presented that resumes the compliance level with the requirements specified in Chapter 4.

5.1 Interaction Model

In Chapter 2.1 the scenario of a device ecology was introduced, being characterized by the presence of *"collections of devices interacting synergistically with one another, with users, and with the Internet"*. It was explained how technologically-enhanced spaces present socio-technical features and it was pointed out that this dissertation only addresses the technical part, which is needed to implement the rest of socio-technical implications. In particular, it focuses on managing interactions among devices, as for example communications and spatial relationships between them and also the I/O capabilities they offer to the user. To this end, several input modalities — touch, touchless and tangible — have been taken into account in order to enable different input modalities. On the other hand, the framework does not cover how users interact with the ecosystem by employing different interaction techniques, neither the technology-mediated human-to-human interactions that belong to the social dimension. Figure 5.1 shows the main components of the framework that are used to model interaction in ubiquitous spaces.

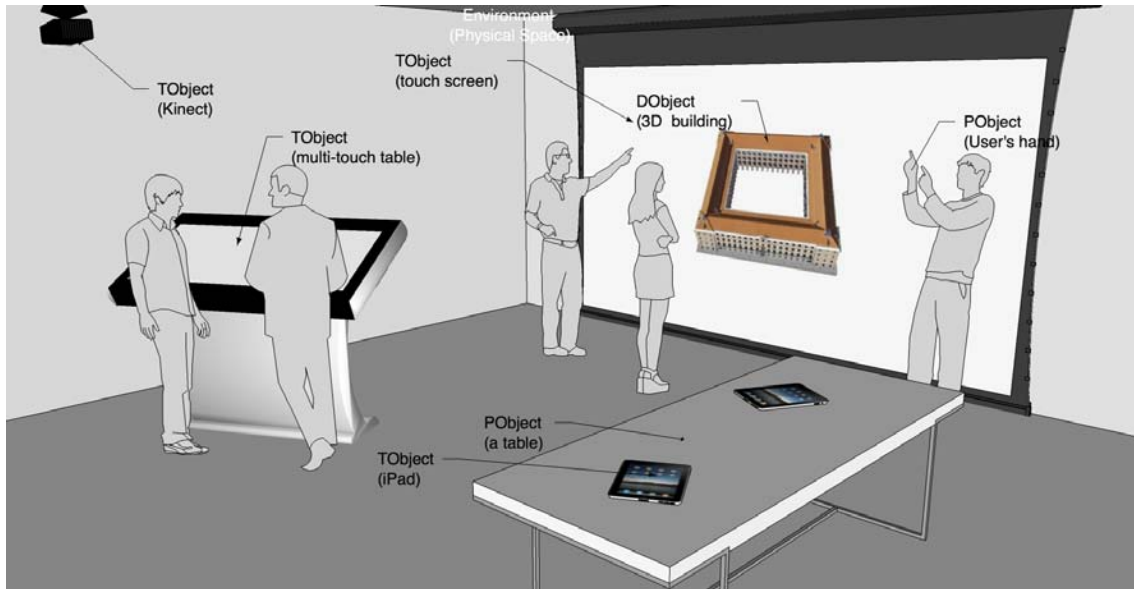


Figure 5.1: Main components of the framework in a real scenario.

The first element is the *Environment* class that represents the 3D space where all the interactions between devices in the ecology take place. Central in the proposed model is the distinction between physical objects (*PObject*) and tangible objects (*TObject*).

A *PObject* is everything inside the *Environment* that shows only physical affordances, for example a normal table, a pen or user's hands or body. They are not input devices but they have physical properties such as dimensions or position that can be exploited to enable interaction inside the *Environment*. A *PObject* is passive and its interaction within the ecology is enabled by *TObject* sensing mechanisms.

A *TObject* is an object with physical affordances enhanced with computational capabilities that can be used as an I/O device, for example a multi-touch interactive table, a Wiimote, an iPad or a Kinect.

The digital counterpart of *PObject* and *TObject* is an object of the digital world (*DObject*), which can be a digital representation of the physical object, thus tightly coupling the physical and the digital worlds, or a digital element with its own properties that can be managed exploiting the input capabilities of a *TObject*. An example of the latter type of *DObject* is a digital image that can be moved, rotated or resized using a touch-enabled display screen. The model proposed by the framework represents a multi-device, ubiquitous extension of the MCRpd framework from Ullmer and Ishii [2000] (Model, Control, Representation-physical and Representation-digital), which in turn is the adaptation to TUI of the Model-View-Controller model (MVC) for standard GUI. In

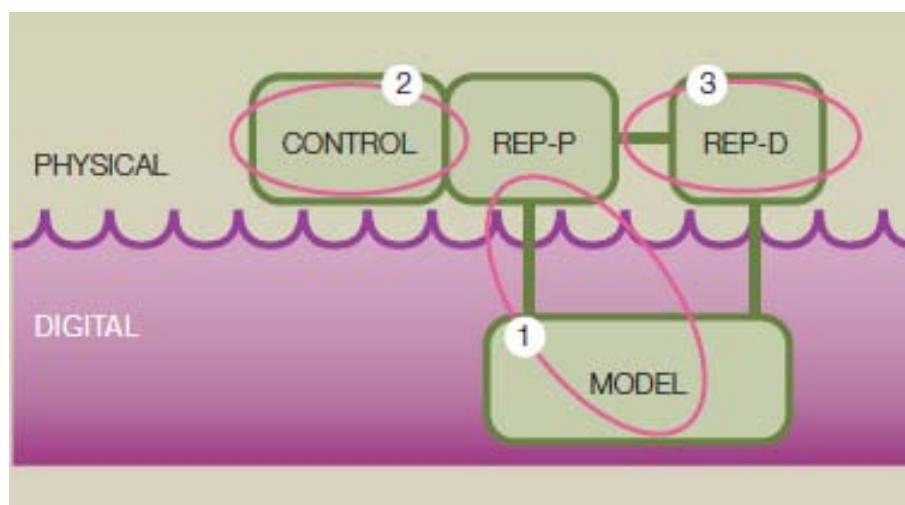


Figure 5.2: MCRpd interaction model for Tangible Interaction. Source [Ullmer and Ishii, 2000].

MCRpd (Figure 5.2):

1. Physical representations (rep-p) are computationally coupled to underlying digital information (Model). The central characteristic of tangible interfaces is the coupling of physical representations to underlying digital information and computational models. A range of digital couplings is possible, such as the coupling of data, operations, and property modifiers. For instance, the metaDESK [Ullmer and



Figure 5.3: Physical icons in metaDESK are coupled with digital information (e.g., the map of the MIT campus).

Ishii, 1997] tabletop system uses a physical icon representing the dome of the MIT university to display the map of the campus (Figure 5.3).

2. Physical representations embody mechanisms for interactive control (Control). The physical representations of TUIs serve simultaneously as interactive physical controls. Tangibles may be physically inert, moving only as directly manipulated by a user's hands. For instance, in the activeDesk [Fitzmaurice et al., 1995] small cubic objects (electronically connected with the system with cables) were employed as controls in a drawing application. In this way, users could control virtual objects not only by fingers, but also through physical elements that acted as handles. Tangibles may also be physically actuated, whether through motor-driven force feedback approaches or by induced approaches. An example of motor-driven approacher are the Height-Adjustable Widgets [Mi and Sugimoto, 2011], that are capable of change their shape according to digital inputs.
3. Physical representations are perceptually coupled to actively mediated digital representations (rep-d). Tangible interfaces rely on a balance between physical and digital representations. Although embodied physical elements play a central, defining role in the representation and control of TUIs, digital representations often mediate much of the dynamic information provided by the underlying computational system. For instance, in the reacTable* [Jorda et al., 2005], while the semantics of physical objects define audio synth objects (e.g., metronomes, filters, modulators, etc.), the topology resulting from the interaction of such physical objects is displayed on the multi-touch tabletop surface (Figure 5.4). Users can interact directly with the digital representation and modify attributes of the physical objects and thus changing their digital behavior.



Figure 5.4: Physical objects and digital representations in the reacTable*.

Based on the MRCpd concept, a similar model for device ecologies has been developed to organize the core *Environment*, *PObject*, *TObject* and *DObject* classes, in a single abstraction. In the original MRCpd model, physical representation is strongly coupled with the digital representation of the system and the tangible objects themselves represent and control the state of the system. Ullmer and Ishii [2000] stated that *"even if the mediating computers, cameras, and projectors [...] are turned off, many aspects of the state of the system are still concretely expressed by the configuration of its physical elements"*. This condition has been relaxed, in the proposed design, because the objective is to model any kind of object in the environment that does not necessarily represent a state of the system. Moreover, MRCpd has been developed only for tangible systems. Even if tangibility of devices and their relationships with the digital world are an important aspect of the framework, tangible interaction only constitutes one of the possible interaction modalities of a technologically-enhanced space. Therefore the framework must be able to model a broader spectrum of configurations in which, for example, tangible objects do represent directly the state of the system, but also scenarios in which they are simply used as I/O devices — imagine the case in which the touch screen of an iPad is used to control a slide presentation projected on a large vertical screen.

Figure 5.5 represents the framework interaction model for device ecologies and in Figure 5.6 on page 99 a portion of the UML class diagram for the model itself is shown.

As introduced above, all the interactions take place inside a space, modeled by the *Environment* class. This class defines the boundaries of the 3D interactive space and provides mechanisms for the tracking of objects (including users) and their spatial relationships in order to enable interactions based on spatial information (e.g., proxemic interactions [Marquardt et al., 2011]). A *TObject* bridges the gap between the real and the digital world: it provides means to get input from the real world such as sensors and physical controls (*TInput*) and produces output of different kind: audio, video or physical

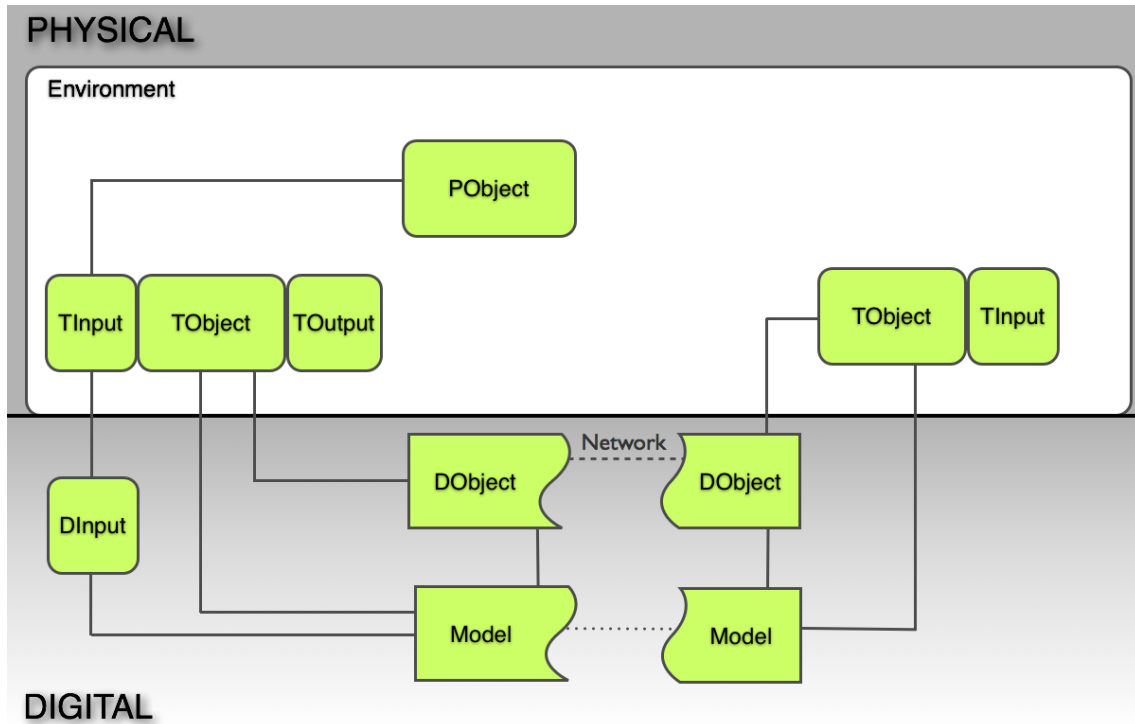


Figure 5.5: The framework interaction model for device ecologies. Core classes (*Environment*, *PObject*, *TObject* and *DObject*) and their interconnections are depicted.

through actuators (*TOutput*). Associated to the *TObject* can also be digital input controls (*DInput*) such as, for instance, virtual buttons, virtual sliders or virtual knobs. *DInput* represents the digital equivalent of the physical controls implemented by *TInput*. Users can directly manipulate a digital object (*DObject*) by means of physical controls and, moreover, the framework allows for another level of input exploiting the graphical elements of the user interface. The *DInput* is independent of the input modality, so that a user can interact with a digital knob by pressing and rotate the finger on a touch-sensitive surface or performing the in-air gesture of rotating the wrist. The *TObject* could also provide a physical knob and have the three different implementations to produce the same action: for instance modulate the frequency of an audio track. *Model* (Figure 5.5, as in MRCpd, is the digital information of the system, but in this case it is distributed over the different devices of the ecology. Devices are interconnected one to the other through a network and, therefore, they can exchange information to generate interactions that are triggered both by their physical state and the state of their digital representations (*DObject*). A *PObject*, which is passive, is tracked by environmental sensors or by the sensors on a *TObject*. For example a user's hand can be tracked by a depth camera placed in a room thus enabling touchless interaction with display screens and, at the same time, this hand can be used to originate touch input on the screen of a tabletop device. In this way, objects in the environment are not only coupled with digital information in

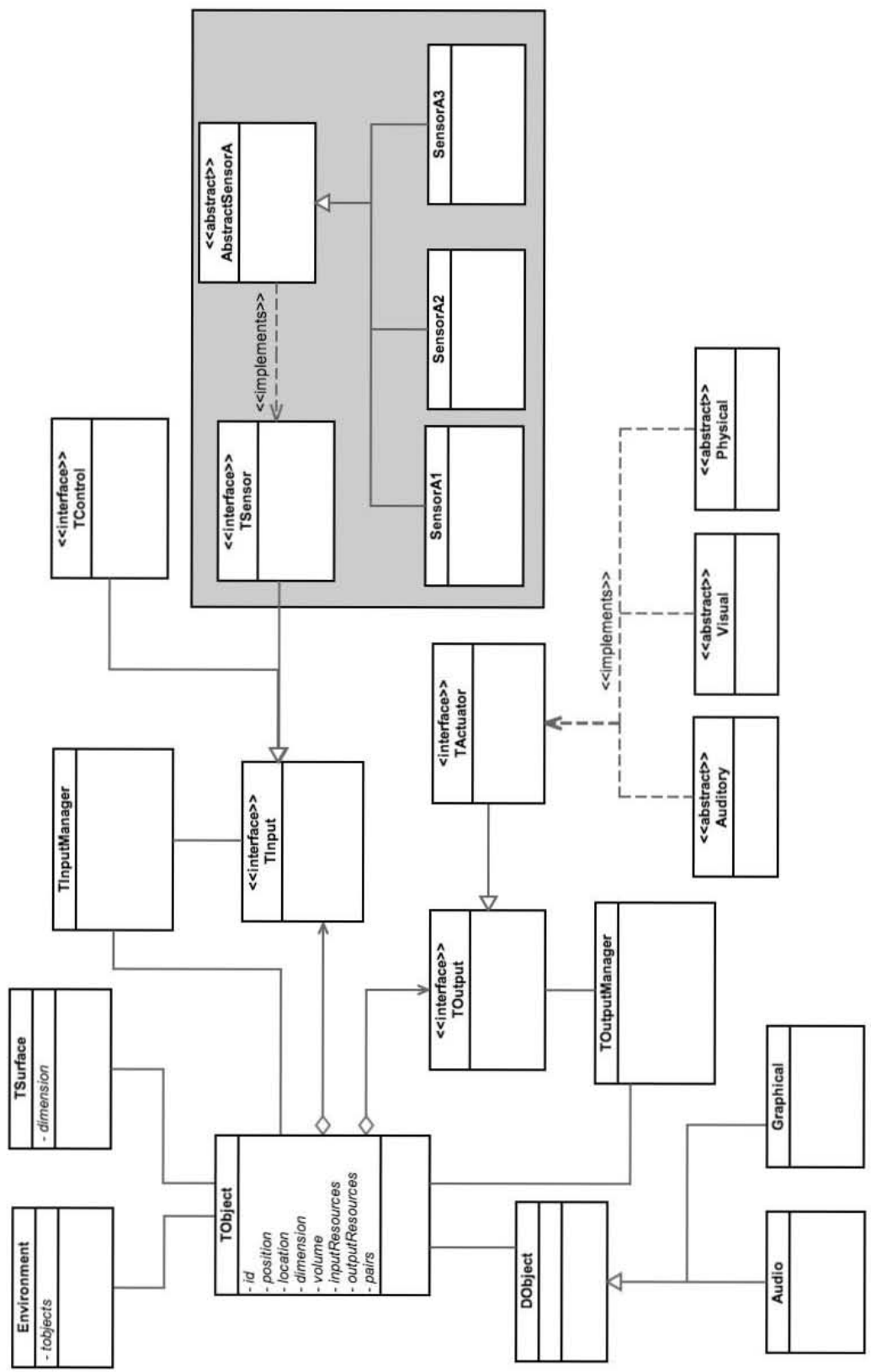


Figure 5.6: UML Class diagram for the proposed interaction model: TObject.

one single spot, but the same physical or digital action can activate physical or digital behavior of other *TObject* in the environment (in Figure 5.5 it is represented by the right side of the diagram). A *TObject*, as well as a *PObject*, can also have a surface (see *TSurface* in Figure 5.6). The difference is that the surface of a *TObject* can have digital capabilities, for instance it can be touch-enabled like in tabletop devices or tablets, while the surface of a *PObject* has only physical properties. A *TSurface* is a 2D version of the *Environment* and generate a nested coordinate space with respect to its father. For instance, a physical puck on the surface of a multi-touch tabletop will have 3D spatial coordinates with respect to the *Environment* and 2D coordinates with respect to the tabletop.

Two examples are now introduced to further clarify the interaction model.

Example 1. Let's consider the scenario depicted in Figure 5.7 and its schematic representation in Figure 5.8.

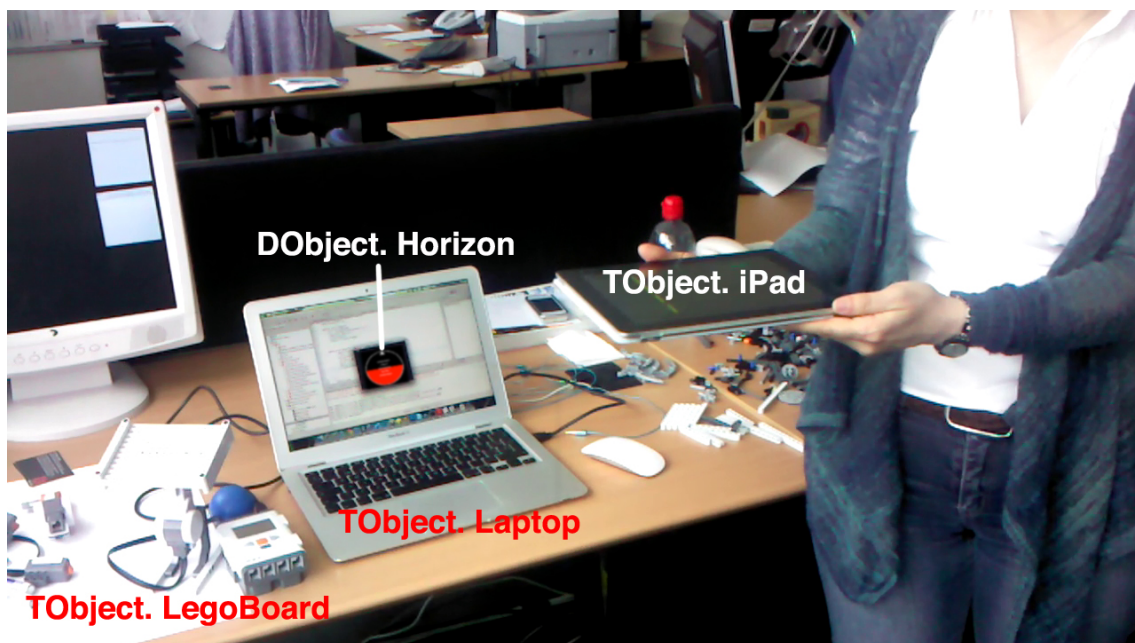


Figure 5.7: Schematic setup of Example 1: the motion sensing capabilities of an iPad (accelerometer) are used to change the inclination of a physical board (implemented with a Lego Mindstorms NXT 2.0 set) and a digital horizon widget displayed on the screen of a laptop device.

The *TObject A*, an iPad is employed to change both the inclination of a physical surface — implemented by the *TObject C* using a Lego Mindstorms NXT 2.0 set¹ — and a digital horizon widget (*DObject D*), which is displayed in the *TObject B*, a laptop. The iPad (*TObject A*) features a motion sensor input, an accelerometer (Accelerometer:TInput

¹<http://mindstorms.lego.com/en-us/default.aspx>

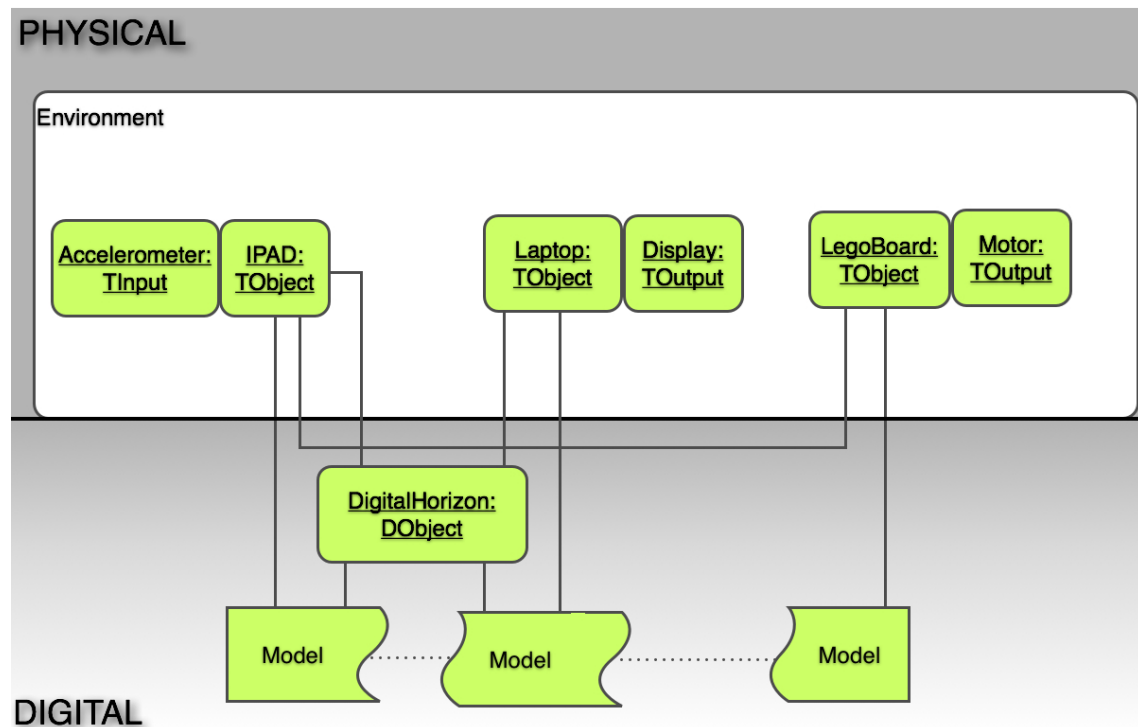


Figure 5.8: Schematic setup of Example 1.

in Figure 5.8). The information of the sensor is processed to get the acceleration of the device along the x-axis and then it is sent to both the laptop (*TObject B*) and Lego Mindstorms board (*TObject C*), which are interconnected with the iPad. On the laptop the digital horizon widget (*DObject D*) is displayed, which is coupled with the iPad. The data received from the accelerometer of the iPad will cause the digital horizon to change its inclination. On the other side, the Lego board is also coupled with the iPad: the acceleration data will cause the motor (*Motor: TOutput*) to move back or forth according to the acceleration force and, therefore, change the inclination of the physical surface.

Example 2. Let's now consider the scenario in Figure 5.9 and its schematic representation in Figure 5.10. When the user touches with his finger the screen of the iPad (*TObject A*) over the virtual button (*Dinput B*) this causes the color of the 3D virtual box (*DObject C*) to change from red to green. The 3D box is displayed on the vertical display screen of a desktop computer (*TObject D*). This example shows how virtual controls can be used to change the behavior or properties of digital objects displayed on different devices. In another scenario in which, for instance, the 3D box would have been displayed directly on the iPad screen, we would have not needed the virtual button and have the color to change by directly touching the box graphical representation.

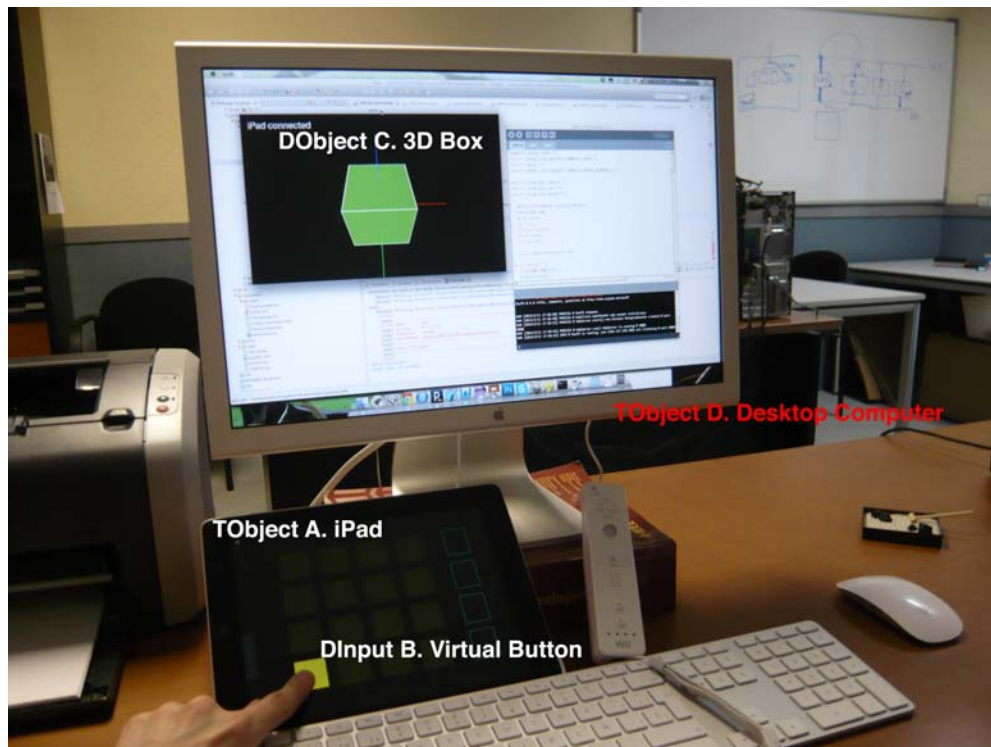


Figure 5.9: Scenario for Example 2: touching the display of the iPad on the virtual button will make the digital 3D box to change its color.

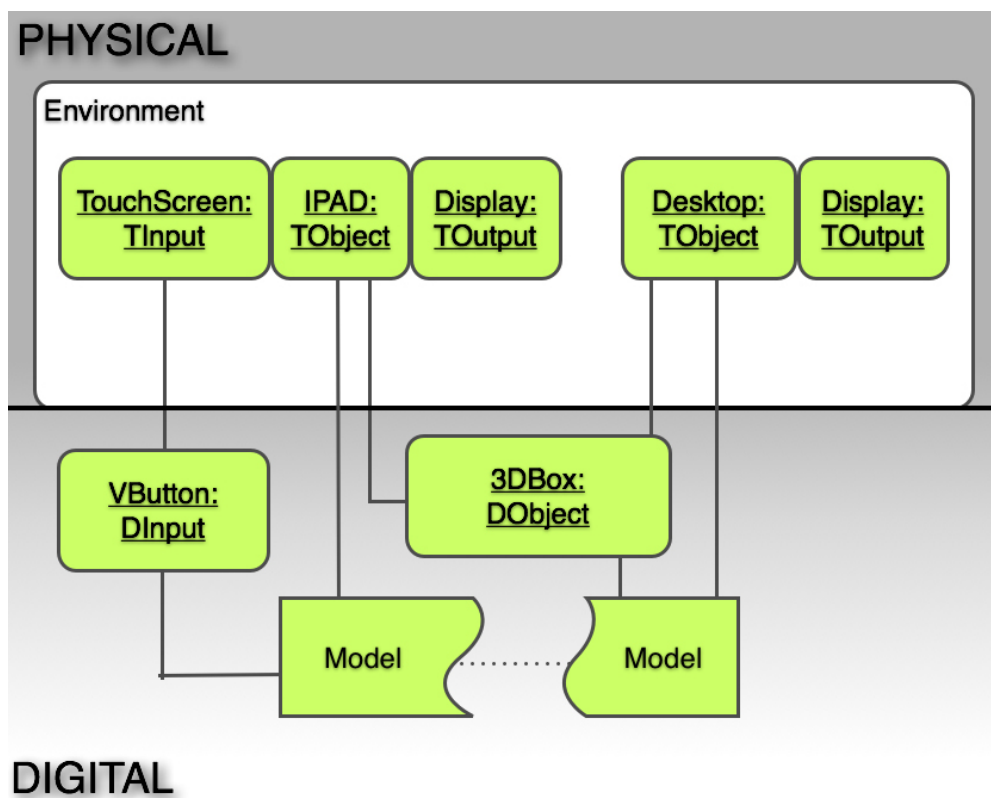


Figure 5.10: Schematic setup of Example 2.

5.2 Architecture

In this section the architecture of the framework is presented, as showed in Figure 5.11, in the next page. The architecture is based on a four layer structure, following the rationale of the architectures proposed by Echtler and Klinker [2008] and Laufs et al. [2010]. Figure 5.11 highlights:

- In **dark blue**, the original contributions of the framework with respect to each layer of the architecture. Each module has been implemented in the prototype. The evaluation is reported in Chapter 6.
- In **light green**, main objects of the model within the rationale of the architecture.
- In **gray**, existing modules from the literature that have been included *as is* in the implementation of the architecture to demonstrate its feasibility.

Hardware Abstraction Layer. This layer allows to abstract from the underlying hardware I/O devices. At this level the *TObjects* that constitute the ecology are defined in terms of their I/O capabilities and raw input data are abstracted according to the data types defined by the framework. Abstract data types are described in details in Section 5.2.1.

Input Transformation Layer. This layer implements mechanisms for the fusion of input and it provides unified data to be processed. Analyzing and interpreting user input are very important for a framework focused on input from different possible sources. In order to have data to be coherent a transformation has to be performed on the low-level data, which is still in device coordinates. For instance, depending on the type of sensor used, a simple scaling or a perspective transformation may be necessary. Moreover, it is important at this level to provide uniform input by fusing data from the same device.

Input Interpretation Layer. In this layer, data is interpreted in order to produce events for the Application layer. Framework events include spatial relationships between *TObjects* and *PObjects* and *TObjects* direct input. In this layer, the term *gesture* is used in the widest possible definition as any kind of physical action that activates an input to the system. For instance, shaking a handheld device can be a gesture but also touching the screen of a multi-touch table or pressing a physical button on the Wiimote. Different middlewares that are compliant to the data types and protocol defined by the framework can be implemented at this stage thus providing the required input interpretation.

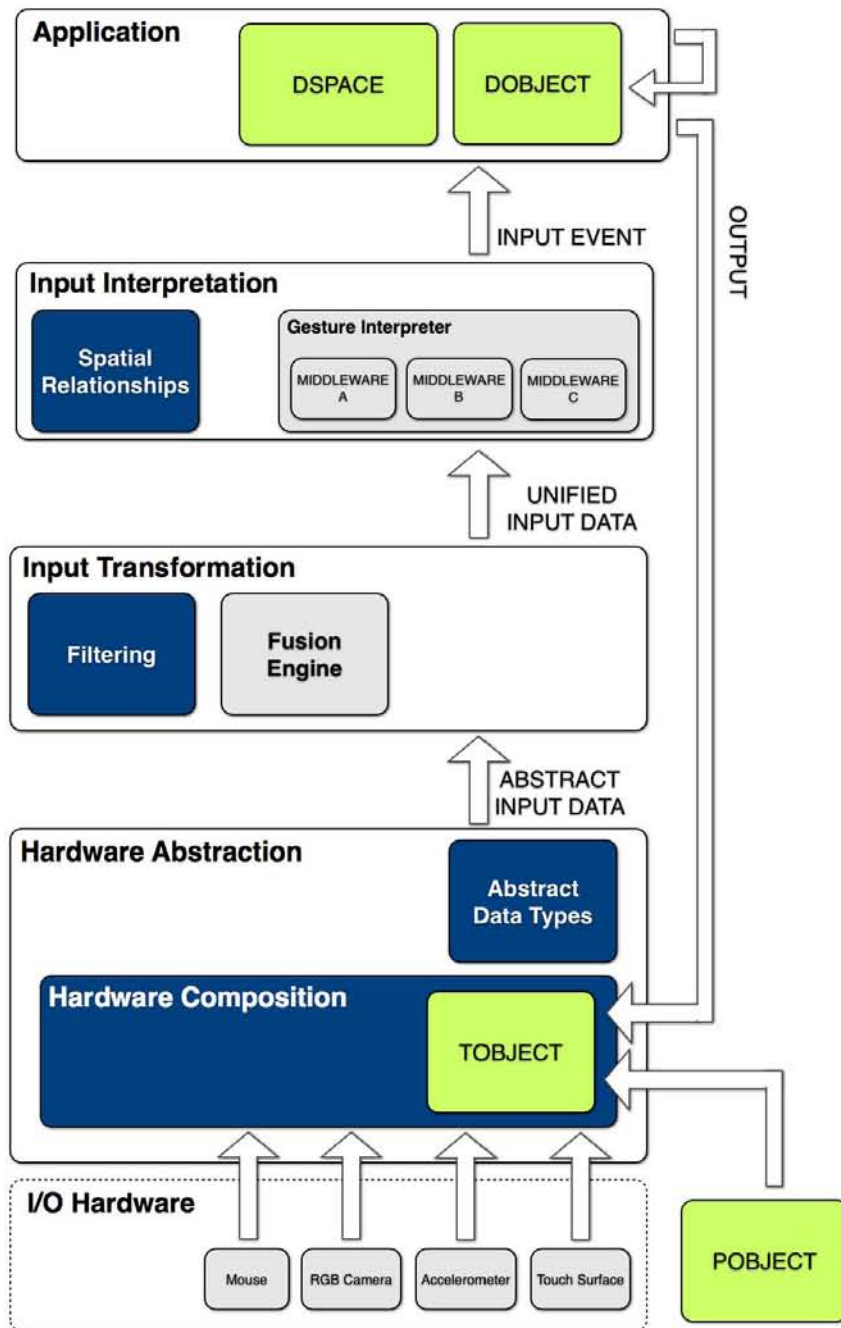


Figure 5.11: The architecture of the framework.

Application Layer. At the application level the user interface is rendered on the devices of the ecology and the input is collected to activate the desired behavior. The input can generate a change in the properties of the digital representations of *TObjects*, for instance a box of a virtual world to change its color or position, but also activate digital controls (the digital counterpart of *TControl* in Figure 5.6), such as virtual buttons.

5.2.1 Hardware Abstraction

Sensors and *actuators* are viewed as a bridge between the real and the digital world, and the framework abstracts from the low-level details of specific components. In this way it can provide unified access to the hardware, independently of its implementation or communication protocols. The framework aim at giving access to sensors and actuators by means of a unified, high-level library that supports the rapid prototyping of interactive systems and the reuse of software components in different applications. Conversely from existing solutions (e.g., [König et al., 2009; Laufs et al., 2010; Wu et al., 2012]), abstraction is provided at atomic level, considering sensors and actuators as elemental components. A *TObject* is therefore a composition of sensors and actuators (as showed in Figure 5.6), which provides the means to generate input (sensors) and produce output in the real world (actuators). For instance, an iPad is a *TObject* that features:

- **Input**, a 197x148mm touch surface, a 3-axis accelerometer, a microphone, one physical button and three physical switches
- **Output**, two speakers (left and right) and a 197x148mm display surface.

Each *TObject* is defined by an XML file in which a list of sensors and actuators is provided, together with the communication capabilities of the object (e.g., bluetooth or network). Instances of *TObjects* are generated automatically from the XML though, in the current implementation they can be generated programmatically from the code. In Figure 5.12 the structure of the XML file corresponding to each *TObject* is showed.

The main tag is *tobject* and it represents the root of the XML document: its attributes are presented in Table 5.1.

Attribute	Type	Description
name	String	The name of the <i>TObject</i> .
id	String	A unique identifier for the <i>TObject</i> used by the framework.

Table 5.1: Attributes of the *tobject* tag.

```

<object name=' ' id=' '>
  <connection type='bluetooth|network' address=' ' port=' ' />
  <position x=' ' y=' ' z=' ' />
  <location x=' ' y=' ' surface=' ' />
  <dimension x=' ' y=' ' z=' ' />
  <tsurface width=' ' height=' ' />
  <tinputs count=' '>
    <tsensor id=' ' type='depth|rgb|accelerometer|touch'>
      <connection type='bluetooth|network' address=' ' port=' ' />
    </tsensor>
    <tcontrol id=' ' type='button'>
      <connection type='bluetooth|network' address=' ' port=' ' />
    </tcontrol>
  </tinputs>
  <toutputs count=' '>
    <tactuators>
      <visual id=' ' type='LED' >
        <connection type='bluetooth|network' address=' ' port=' ' />
      </visual>
      <auditory id=' ' type='Speaker'>
        <connection type='bluetooth|network' address=' ' port=' ' />
      </auditory>
      <physical id=' ' type='servo'>
        <connection type='bluetooth|network' address=' ' port=' ' />
      </physical>
    </tactuators>
  </toutputs>
  <pairs>
    <pair idto=' '>
      <connection type='bluetooth|network' address=' ' port=' ' />
    </pair>
  </pairs>
</object>

```

Figure 5.12: The structure of the XML file corresponding to a *TObject*.

The *connection* tag is used to abstract object connection channels. In this way developers do not need to care how devices communicate. At present time two methods are implemented, bluetooth and network. This tag can be used inside the main *tobject* tag, but also each component of the *TObject* can present its own connection interface. This functionality of the framework has been implemented to provide a tool for the rapid setup of devices with sensors and actuators from different sources. For instance, a designer might want to create a setup with different interconnected tablet devices, such as an iPad. They also want to extend iPad interactive capabilities with an ambient display let's say, in this simple scenario, a LED — the simplest actuator to provide visual output. They can build their LED actuator with the Arduino platform, for instance, and provide the component with a bluetooth interface. The main *TObject*, which in this case is made by the combination of all the iPad sensors and actuators plus the new LED component, will expose the wifi network connection from the iPad to the outside, in order to communicate (send and receive events) with other devices in the ecology and use the bluetooth connection of the LED component for internal communications. From the outside, for instance another device in the ecology, access to single components of the *TObject* is mediated by the *TObject* itself that receive the event that activate, in this case, the LED actuator. The other devices do not need to be aware of internal communication channels. Moreover, by defining internal connections in the XML file allows developer to access any component of the *TObject* using the same interface, regardless of the implementation. *TObjects* handle internal connection automatically and developers do not need to instantiate them. In Table 5.2 attributes of the tag *connection*.

Attribute	Type	Description
type	String	The connection type. Implemented types are <i>network</i> or <i>bluetooth</i> .
address	String	The address of the connection (bluetooth or IP).
port	Int	The port of the connection. Only for network connection.

Table 5.2: Attributes of the *connection* tag.

Each *TObject* has a position (tag *position*), specified in terms of x, y and z coordinates of the *Environment* and may optionally has a location (tag *location*), which is identified by the x and y coordinates of a *TObject* with respect to a surface in the environment. The *dimension* tag represents the 3D bounding box that contains the *TObject*. Optionally, the *TObject* can also have a surface that can be used as a spatial reference frame for nested objects. In this implementation only planar surfaces are supported and no references have been implemented regarding its spatial position inside the *TObject*. As showed in Figure 5.12, inputs and outputs are defined in terms of:

- **TInputs**, which can be sensors or physical controls.
- **TOutputs**, which can be different type of actuators providing visual, auditory or physical feedback.

Both tinputs and toutputs components, as reported above, may present a *connection* tag, that indicates the type of connection for internal communications between components of the same *TObject*. At this stage of framework development the following components have been implemented:

- **Sensors**: depth and RGB camera of the Kinect, 3-axis accelerometer of the iPad, Wiimote and Arduino, touch screen of the iPad.
- **Controls**: Physical buttons of the Wiimote
- **Actuators**: LED for the Wiimote and Arduino, Wiimote speaker, Wiimote and Lego MindStorm servo motor.

Finally, each *TObject* has a list of pairs (tag *pairs*) that indicates the other devices in the ecology that are connected with it and are able to detect spatial relationships. The XML file that describe a *PObject* in the environment is limited to the physical tags of *position*, *location* and *dimension*, since *PObjects* have no digital interactive capabilities and are used only to establish spatial relationships in the environment.

Client/Server architecture. Each *TObject* is modeled according to a Client/Server architecture (Figure 5.13). Once connections between *TObjects* are established, according to the XML configuration file, a *TObject* can register to listen for events produced by other *TObjects* in its list of pairings. This mechanism implements the event-driven pattern of the framework (see Section 5.2.5). Each *TObject* in the environment is connected to the environment server that maintain the list of all the interactive and static objects tracked in the space and manage the establishment of connection between them and initialize handlers to monitor relationships between objects.

Abstract Data Types. As mentioned in Chapter 1 in order to unify heterogenous I/O devices in a unique framework and define middlewares for agnostic input management, a common language is needed to allow data exchange among components independently of their implementation. The foundation of such unification has been given, in this work, by the generalization of input devices from Wallace [1976], which as also been used for the same purpose in other projects such as the Squidy library [König et al., 2009]. Wallace [1976] tried to specify "*a set of virtual devices which appear to be complete and "machine independent", and to simultaneously allow reasonable preservation of device and program power, efficiency, and flexibility*". In particular he identified a five distinct virtual devices:

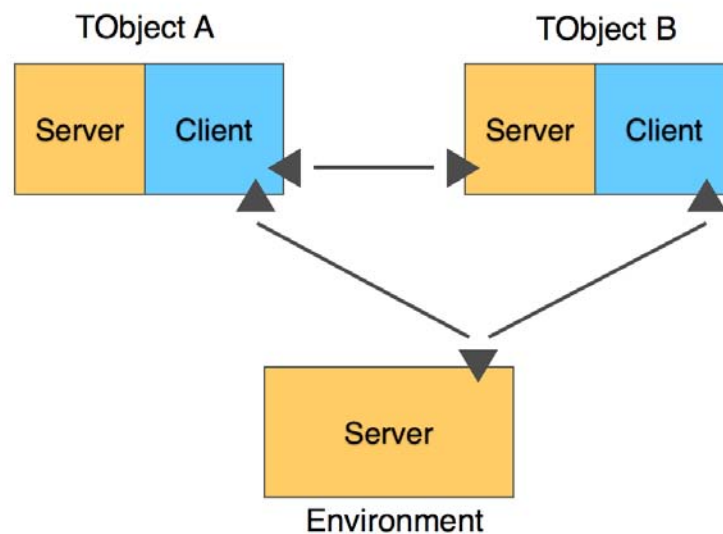


Figure 5.13: Client/Server architecture of *TObjects*.

- **Valuator.** It is a one-dimensional value within the real vector space. For example, rotating a potentiometer sends discrete values whereas the vector increases when rotating clockwise and decreases when rotating counter-clockwise. A linear prototype would be a linear slider component.
- **Locator.** It provides position and orientation in a vector space. This can be a two dimensional vector in its primitive representation (e.g. a location of a touch point on a planar display). Furthermore, multi-dimensional values are possible such as a 6DOF vector including x-, y-, and z-axis value plus the three rotational values pitch, yaw, and roll around the 3d location.
- **Button.** It represents boolean states and the like. Values such as true/false, 0/1, and on/off are potential values of a button device . The most popular device equipped with button devices is the computer mouse. Several mouse buttons response to user input such as “press” or “release”.
- **Keyboard.** A keyboard device is a sequence of ASCII characters. Each sequence can be terminated by a particular character sequence such as the zero-byte in a null-terminated string. The prototype of a keyboard device is the keyboard as input device.
- **Pick.** It was developed in response to users’ need to point at displayed objects created during execution. A sampling of a pick produces a reference to a graphic object currently being pointed at. The prototype pick is the stylus IR pen.

As a consequence of the work of Wallace [1976] it is possible to exploit the semantics of virtual devices to unify heterogeneous devices, toolkits and frameworks into a generalization of various kinds of input and output data to a hierarchy of well-defined data types. The major benefit of abstract data types is that a unique definition of devices input and output data allows the implementation of input agnostic middleware as reported in *Input Interpretation* section. The data types offer a high-level encapsulation of atomic data types defined by particular programming languages and thus provide semantically bundled data container that make easier for the developer the routing of low-level data, which can be error-prone. Moreover, it offer the building block for the cration of a data-flow visual programming language, as reported in future works (see Chapter 7). The data hierarchy of the framework is based on the previously introduced primitive graphic devices and the hierarchy proposed by König et al. [2009]. The original contribution of Wallace [1976] has been adapted and extended in order to model the type of data produced by actual ubiquitous devices and, hopefully, to be powerful enough to model future device implementations. The data model is reported in Table 5.3. The *Value* data type, which

Abstract Data Type	Dimensions	Prototype Device
Value	1D	Potentiometer
	3D	3-axis accelerometer
Location	2D	Touch surface
	3D	3D pointer
	6D	Wiimote
Choice		Button
String	1D	RGB camera
	2D	RGB + Depth camera
Pick		Mouse pointer

Table 5.3: Abstract data type hierarchy. The first column presents the abstract data type, the second column an example of a multi-dimensional extension of the data type and the third column an example of a device generating such input.

originally was used to model only one-dimensional discrete data, has been extended to multiple dimensions in order to, for instance, model 3D values such as the input provided by a 3-axis accelerometer. *Location* is mainly used to represent the location of objects (both *TObjects* and *PObjects* in the environment space. Through *Location* data spatial relationships are computed as showed in Section 5.2.3. Original *Button* device takes the form of *Choice* data in the framework, to model binary input. *Keyboard* has been replaced by *String* data type. A string is a sequence of characters or any kind of data and, therefore, is exploited in the framework to model any input device that generates an input stream of any kind. It can be, for instance, a RGB camera producing a stream of bytes representing an image or a RGB plus Depth camera, such as the one provided by the Microsoft Kinect, which generates a 2D stream of RGB data plus the depth information associated to any element of the RGB stream. The *Pick* data is implemented as a

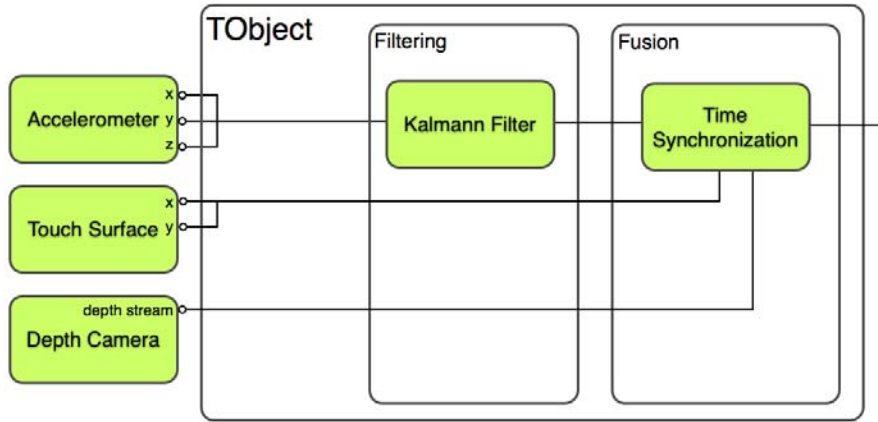
reference to an object being selected. Pick is not directly used in current implementation but it can be useful in a variety of future situations. For example, *"the item picked is automatically visually reinforced, as by brightening or flashing"* Wallace [1976], thus it can be useful to implement visual feedbacks of a selection. Although Pick data type can be implemented using Location data, we believe it is useful to have reference data to be logically separated from location data. In the same way, the framework provides digital controls, along with physical ones, and digital objects (*DObject*), the same data type hierarchy is also used for the information generated by such virtual elements. For instance, a virtual button will generate data of the same *Choice* type of a physical button. Real and virtual input is therefore unified into a common data model and architecture.

Communication Protocol. *TObjects* events, such as the input produced by sensors and controls, are reported via the TUIO v2.0 protocol. TUIO CTL messages are used to this end since, as described by the specifications²: *"The CTL message can be used to transmit additional control dimensions that can be associated to an existing component instance, such as a token with an incorporated pressure sensor or for example. The variable length list of floating-point values, encodes each individual control dimension in the normalized range from -1.0f ... 1.0f. An embedded three-axis accelerometer for example can be encoded using a CTL message with three control dimensions, providing three floating-point attributes after the associated session ID"*. The use of the protocol is completely transparent to the developer, who only needs to specify the sensor or control id in the XML descriptor. Also TUIO DAT messages are used to send input events in case of more complex data, as for example the fusion of different sensor input for the same *TObject*.

5.2.2 Input Transformation

In this layer, data from the underlying hardware is processed before it reaches the client side of other *TObjects* that are connected. Here, data processor can be implemented as proposed by Israel et al. [2011]. In the current implementation of the framework, for instance, a Kalmann filter [Welch and Bishop, 1995] has been implemented to perform smoothing of data. The fusion engine is an important module of any multimodal system for it is in charge of managing input from different source in a unique frame. For instance, time synchronicity among input modalities is particularly problematic since the interpretation might vary depending on the time at which modalities are used. We do not directly address the problem of input fusion in this dissertation, which is object of active research in multimodal human-computer interaction. Only a simple time synchronization mechanism has been implemented based on the timestamp of the event generated by each sensor of a *TObject*. Figure 5.14 depicts the actual implementation. Depending on the

²<http://www.tuio.org/?tuio20>

Figure 5.14: Input Transformation mechanisms in a *TObject*.

needs of the *Input Interpretation* level, input can be grouped into a set of heterogenous data, depending on their timestamp, or used alone. For instance, imagine to have a *TObject* that features an accelerometer. If at the *Application* level we are interested in getting the accelerometer information to have a digital object changing its inclination on the X axis (pitch), we can directly process accelerometer data alone. Fusion mechanisms at *Application* level, for data from different *TObjects* have not been fully implemented. Again, a fixed time synchronization mechanism is used based on a window of 25 frame per second. In this way it is possible to capture events such as: change the pitch of the *DObject A* if and only if the *TObject B* is at less than 1m away from the *PObject C*, using the input from its embedded accelerometer. Distance information can be captured from an environmental sensor and the spatial relationship event is computed as reported in the next section.

5.2.3 Input Interpretation

In this level generic input from *TObjects* is interpreted in order to generate more complex input that include spatial relations between objects in the environment and gestural commands. Spatial relationships allow to implement proxemic interactions as depicted by Marquardt et al. [2011]. In the current implementation of the framework, the spatial relationships supported are reported in Table 5.4, grouped in the three categories of *Topological*, *Directional* and *Distance* according to Freeman [1975]. Such relationships are computed by using both information from the environmental sensors and the sensor on each *TObject*. For instance, two Wiimote have been used to track objects in a 3D space. Objects were equipped with two LED to enable tracking. People have been tracked using a Microsoft Kinect as well as in the case of hand movements.

Category	Relation	Description
Topological	contains(a,b)	$a \cap b = b$
	within(a,b)	$a \cup b = a$
	touches(a,b)	a is touching b , that is $distance(a, b) \sim 0$
Directional	left(a,b), right, onThe-Back, inFront, super, sub	Specify where the object a is located outside of the reference objects b .
	pointsToward(a,b)	True if a is pointing in the direction of b , that is if the infinite pointing ray from a intersects b .
Distance	$distance(a,b) = distance(b,a)$	Distance between a and b

Table 5.4: Spatial relationships between two entities a and b .

Interpretation of gesture from input data has not been implemented in the framework, but the architecture allows for different middlewares to be used at this level to generate gestural interaction event. The objective of the *gesture interpretation* module in this layer is to allow the creation of device agnostic interaction, that is the input event is generated independently of the underlying hardware. The implementation of such middlewares that leverage on the abstract data type model has been left for future work.

5.2.4 Application

At the *Application* level different digital entities have been implemented that represent the virtual complement of physical objects. The main *Environment* class is implemented through the *DSpace* class, that is virtual 2D or 3D space where digital objects exist. For example in a tabletop device a 2D *DSpace* can be defined that represents the digital surface of the table and elements of the interface are displayed. Two classes of digital entities that populate the digital world are provided by the framework, which have already been described in this chapter:

- **Digital Objects** (*DObject*) are 2D or 3D objects of the user interface. They can be directly linked to a *TObject* so that any physical input can correspond to a digital behavior of the *DObject*. At the current stage basic objects have been implemented such as primitive 2D and 3D shapes. Moreover, in the case of 2D visualization, a number of multimedia objects are supported, such as images, videos or interactive maps. Such as real objects, digital objects can react to spatial relationships. The same rationale proposed for spatial relationships between *TObjects* has been reproduced for *DObjects* in the virtual world. Just as these spatial relationships among *TObjects* produce input events that affect the behavior of other *TObjects* (e.g., using actuators) or their digital representation, spatial relationships among *DObjects* can cause output both in the digital and real world.

- **Digital Inputs** (*DInput*). Just like real world input devices (sensors and controls), virtual input devices have been implemented that take the form of *DControls* that is a subclass of *DInput*. A *DControl* is the virtual representation of a physical control device, such as a button or a slider. A *DControl* is activated by the input from a *TObject* and, as explained in the previous section, allows to build hardware agnostic input mechanisms. Example 2 in Section 5.1 presents a scenario that illustrates how virtual input is implemented and used for interaction.

5.2.5 Event propagation

The framework provides four levels of events:

- The events produced by sensors and controls of a *TObject*. These events are generated at the *Input Transformation* layer.
- The events produced by *Spatial Relationships* or *Gesture Interpretation* modules in the *Input Interpretation* layer.
- The events produced by digital controls (*DControl*). Such events are fired by input events of the two previous types.
- The events produced by spatial relationships between *DObjects*. These events can be fired by any of the three previous types.

5.2.6 Implementation

The framework has been implemented in form of a development API offered via an object-oriented Processing and Processing.js library. The programming language used is, therefore, Java and in case of iOS devices, Javascript. Processing sketches run on any *TObject* that needs to display the user interface and process the input from connected *TObjects*. Essentially, the API allows developer to define the entities that constitute the ubiquitous space and their relationships. In the next sections the implementation at each level of the architecture is presented in details and examples are provided to show how the API is used to program desired behaviors.

In order to clarify the implementation of the framework, a working example is now introduced.

A sample scenario. Andrea wants to use the framework API to develop the following interactive scenario: he wants to use the motion sensing capabilities (accelerometer) of an iPad (*TObject A*) to rotate a virtual cube displayed on a vertical digital screen surface

on the wall (*TObject B*) and he wants this happens only when the Wiimote is at less then 1 meter from the surface. In this case the cube is colored in green and reacts to Wiimote input, otherwise it will be colored red and do not receive any input command. In Figure 5.15 parts of the Processing source code to is showed that runs on the machine connected to the digital screen display (the *TObject B*).


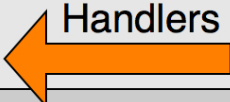
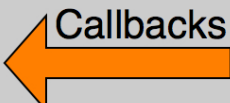
<pre> 1. init(); 2. Environment env = new Environment("163.117.138.191", 8001); 3. register(env); 4. TObject ipad = env.getTObject("ipad_id"); 5. connect(ipad); 6. SpaceRelationship r1 = new SpaceRelationship(this, ipad); </pre>	 <p>Initialize</p>
<pre> 7. addSpaceRelationshipHandler(this, r1, Class.forName("DistanceUpdateEvent")); 8. addObjectInputHandler(this, ipad, Class.forName("AccelerometerUpdateEvent")); </pre>	 <p>Handlers</p>
<pre> 9. handleDistanceEvent(DistanceUpdateEvent evt){ 10. if (evt.toObjectID().equals(ipad.id) && evt.getDistance() < 1000){ 11. [... set a flag to TRUE to enable accelaration events and set the color of the 3D box to green...] 12. } else{ 13. [... set a flag to FALSE and set the color of the 3D box to red...] 14. } 15. } </pre>	 <p>Callbacks</p>
<pre> 16. handleAccelerometerEvent(AccelerometerUpdateEvent evt){ 17. if(evt.getSourceID().equals(ipad.id) && [... the flag is TRUE ...]){ 18. Value3D accelerationInput = evt.getData(); 19. [... access the acceleration data to rotate the cube ...] 20. } 21. } </pre>	

Figure 5.15: Part of the source code for the scenario.

Let's assume that the environment server is running, the XML description of all the involved entities — the two *TObjects A and B* — has been loaded and the connection with A and the server has been already established. B extends the *TObject* class. The *init()* method initialize B's client and server modules (line 1). Andrea will then write the code to connect B to the server (line 2-3): in this way the server will also start to track B position in the physical space. Then, with lines 4-5, the connection with the iPad will be made. Line 6 creates the object to define a spatial relationship between entities, in this case *TObject A* and *TObject B*. In lines 7-8 the handler to manage event message are established: the first one (line 7) will manage distance events while the second one (line 8) will manage accelerometer events fired by *TObject A*, which has been instantiated in line 4. Lines 9-21 define the callbacks for the event handlers: lines 9-15 describe the behavior of *TObject B* when receives distance event and lines 16-21 acceleration event. For instance line 10 checks if the distance is less then 1 meter (1000 millimeters) and then triggers the associated behavior as described in this scenario. The same happens for the acceleration events callbacks: first it checks the source of the event (line 17)

because the same *TObject* could have registered for the same event type from different sources. In line 18 the data from the sensor is got according to the abstract data type (an accelerometer returns *Value3D* data objects).

5.3 Compliance with requirements

In Table 5.5 requirements for the framework, identified in the previous chapter are reported together with a description of how they have been addressed in the design of the framework and its implementation or if their inclusion has been left for future works. The following color code has been used for compliance levels: (a) green means fully addressed, (b) yellow: partially addressed and, (c) red: left for future works.

Category	Requirement	Level	Description
Input/Output Hardware	Support Heterogeneous Input/Output Hardware		Though the framework has been designed to support multi-touch, tangible, touchless and traditional devices, including sensors and input from visual markers, current implementation is limited to a small set of sensors and devices, as reported in Section 5.2.1. The modular architecture and the abstraction provided by the <i>Hardware Abstraction Layer</i> and the data type hierarchy are thought to allow the implementation of any current device.
Interaction Modalities	Support Heterogeneous Interaction Modalities		The support of different interaction modalities has not yet been fully implemented though the architecture allow for the seamless integration of middleware to manage input from heterogeneous devices. Multi-touch interaction is supported only to basic touch inputs and gestures are not allowed in the current implementation.
Interactive Space	Spatial Awareness		Entities (devices and users) are identified and tracked in order to fully support spatially-based interaction. Entities are aware of their location and the location of others.
	Multi-Surface Environment		Ubiquitous environments are characterized by the presence of multiple surfaces that span across different objects and devices. Surfaces can therefore provide digital behavior or not. Both cases are supported by the framework: in the case of interactive surfaces they can be used to visualize digital content and also as a means for input through touch or tangible interaction. Physical surfaces are exploited in order to enable proxemic interaction.
Architectural Treats	Distributed Architecture		The framework provide a distributed architecture that allows for full-duplex communication between connected devices.

	Common Communication Protocol		Device interoperability is achieved by exploiting the TUIO protocol for exchanging messages between entities.
	Expandible / Extensible Architecture		The abstract data model allows to implement new composite devices. The event driven architecture has been developed to allow users to define specialized input event that extend predefined classes.
	Device Abstraction		Device Abstraction is achieved in the <i>Hardware Abstraction</i> layer by means of the mechanisms to define interactive objects as a composition of I/O components and the abstract data type hierarchy.
	Manage Different Data Streams		Data streams from different sensors are managed from the perspective of single devices and from the whole ecology. Timestamps are associated to I/O data and a simple fusion mechanism has been implemented. Nevertheless, better approach to multimodal data fusion need to be explored.
	Agnosticism of Legacy Middlewares		The framework provides a layer that allows for middleware from third party to interpret input event independently of the underlying hardware. However only a simple example that allows touch input from the data produced by a depth sensor has been implemented.
Developing / Coding	Common Programming Language		The framework has been implemented using Java and Javascript.
	Familiar Development Platform		The framework has been implemented in the Processing environment, which is widespread extended in the community of multimedia designers and coders.
	Low Viscosity of the code		Users can easily change interactive devices implementation without need to make substantial changes to the source code. Figure 5.15 illustrates how with a few lines of code an interaction between two objects in the environment can be easily setup.
	Hide Low-Level Coding Details		Interactive objects are automatically instantiated by parsing the XML configuration file. Moreover users do not need to take care of the connection between objects.
	Provide Programming Alternatives		The framework allows the programming via API, hard-coding and configuration file. Visual programming tools are not supported in current implementation but one of the objective for future works is to provide users with visual tools that ease the development process.
Application / User Interface	Real (e.g., haptic)/Digital(e.g., visual) User Feedback		The framework provides mechanisms to generate output according to different channels, such as physical haptic feedback via actuators on tangible objects or digital feedback by means of virtual objects.

	Cross-Device UI	Green	The framework, through its distributed event-driven architecture, allows the user interface to be divided amongst different devices.
	Storage and Re-play of Interactive Sessions	Red	No mechanisms to store interaction information has been designed.
	Easy configuration of Input Devices	Green	The configuration of devices as tangible objects is made via a XML configuration file that describe the interactive properties of the object.
	UI Widgets	Yellow	Digital objects have been implemented that can act as digital controller, such as virtual buttons or sliders. However more element of the graphical interface need to be added.

Table 5.5: Requirements compliance matrix. Green: fully addressed. Yellow: partially addressed. Red: future works.

5.4 Summary

In this chapter the design and implementation of the framework have been presented, focusing on the interaction model, the four layer architecture and implementation details. The interaction model provides a scheme for interactions between objects in a responsive environment. A distinction between physical and tangible objects has been made, being the latter enhanced with digital capabilities. Entities of the environment — including the users — form an ecology where each entity affects the behavior of others. Input events can be physical (e.g., pressing a button) or digital (e.g., two virtual objects become closer) as well as the output: an example of physical output can be a motor making an object vibrating while digital output interests any digital property of a virtual object (e.g., shape, color, location in the virtual space, etc.). Examples have been presented throughout the chapter that show how the interaction model works in real situations. The architecture has been defined following the rationale of previous proposal in the state of the art, in particular focusing on the work of Laufs et al. [2010] and Echtler and Klinker [2008]. The four layers of the architecture — *Hardware Abstraction*, *Input Transformation*, *Input Interpretation* and *Application* — have been described that allow to implement the interaction model according to the requirements from Chapter 4. The framework has been designed addressing the conditions expressed by each requirement; nevertheless, many requirements have not been fully targeted and one of them — Storage and Replay of Interactive Sessions — has not been included in current implementation, which leaves room for further work. Neither visual tools have been implemented that have been demonstrated to be effective for rapid prototyping [Klemmer et al., 2004; König et al., 2009; Marquardt et al., 2011]. The development of a unique, comprehensive

framework for interaction in ubiquitous system turned out to be an ambitious task and, at this stage, the efforts converged on the definition of the model, architecture and the specification of functional APIs that relieve developer from the burden of deal with low-level details. In the next chapter, a user evaluation is presented to demonstrate that the proposed solution make easy the development process in terms of programming time.

6

Evaluation

That what does not kill us, makes us stronger.

FREDERICH NIETZSCHE, GERMAN PHILOSOPHER

IN this chapter the evaluation of the framework is presented, within the conceptual scheme for user interface evaluation defined by Myers et al. [2000]. The evaluation has been carried out using: (a) a use case and, (b) a user test. Firstly, the methodology from Myers et al. [2000] is described, focusing on the dimensions established and how they apply to the framework. Then, the use case of a digitally-augmented product shelf is presented to demonstrate that through the software libraries it is possible to develop ubiquitous interaction systems. Lastly, the user study sheds light to the usability of the APIs to develop rich experiences, in a fast and simple way, by integrating different input devices within a comprehensive architecture. In particular it demonstrates that the framework positively affects the development time of ubiquitous interactive systems.

6.1 Evaluation Background

Myers et al. [2000] asserted that software toolkits assist developers in the design and implementation of the user interface. For instance, at the beginning of the nineties there was no toolkit support and 48% of the programming effort was dedicated to the coding of the user interfaces [Myers and Rosson, 1992]. The advent of software toolkits has surely had a tremendous impact on development—programmers needed less code for the user interface that, therefore, could be developed more quickly and with less errors. In particular, Myers and Rosson [1992] demonstrated that user interface development kits could reduce the development time by at least a factor of four. During the last twenty years, the research on software tools has allowed to build more and more refined development environments and, nowadays, it can be said that any application is built using a Software Development Kit (SDK) or an Integrated Development Environment

(IDE) or some sort of toolkit that exposes API functionalities or a palette of components for the rapid generation of the graphical user interface. Reducing the time to create the user interface directly means to be able to produce more prototypes and, therefore, more design iterations, which is crucial to achieve high-quality interactive systems [Nielsen and Hackos, 1993]. Given the importance of software tools in the development of user interfaces, Myers et al. [2000] reviewed the literature of past user interface tools and, by analyzing cases of success and failure, they were able to extract a set of dimensions that frame the space for past and future tools. They reported that their dimensions can be used as a lesson—they are a guide for developers of future tools that may help to avoid pitfalls and build a successful tool. They also pointed out that framing toolkit research with a set of dimensions is particularly important in present days in which we are experimenting rapid technological changes that are affecting the way users perceive and interact with computers. Following Weiser [1991]'s vision, they also pictured a future in which the desktop box will be opened and an *"increasing diversity of user interfaces on an increasing diversity of computerized devices"* will be created. Myers et al. [2000] wrote their article more than ten years ago when we were just witnessed the dawn of the ubiquitous computing era and now that technological enhancements are materializing Weiser's ideas, software support is even more essential in order to face the demands of novel interactive environments. For instance, the new ubiquitous interactive scenario is made of devices with different form factors, display dimensions, input methods and context-awareness capabilities. This might lead to the study of techniques for the specification of device-independent interaction: the system should be able to choose appropriate interaction techniques, by taking into account the I/O capabilities of the device, the users' preferences and the needs of the application. Myers et al. [2000] stated that *"it is especially important to explicitly consider the effects our tools will have on what we can and will build, and to create new tools that have the properties needed to meet a new generation of demands. There are many examples that show that tools have significant impact on the styles of interfaces that are created. For example, in the World-Wide Web, it is actually easier to use pictures as buttons rather than to use "real" button widgets. Therefore, designers created elaborate, animated user interfaces with rich visual design and high production values"*. The same vision is shared by Greenberg [2007] who advocated a *toolkit culture* to promote creativity amongst developers. Toolkits are considered as a common language—they allow designers to place themselves atop the babel of programming languages, input devices and interaction modalities thus focusing on the design of the interactive system. Furthermore, toolkits are not just a mean to reduce the complexity of user interface and implementation, they are not just a way to do the same things but quickly. Rather, they change developers perception and understanding of the interactive environment and give the possibility to build novel and meaningful configurations.

6.1.1 The five themes

The five dimensions for the evaluation of successful toolkits for the development of user interfaces are reported here, as depicted by [Myers et al., 2000]:

- **The parts of the user interface that are addressed:** *"The tools that succeeded helped (just) where they were needed"*. They observed that most successful tools do not follow the *swiss army knife* model to provide a large variety of general purpose functionalities but instead they focus on one or a very restricted set of issues of the user interface, which they address comprehensively. In the case of the the proposed framework, it attempts to cope with the interoperability challenge by provide device abstraction and a common data type for the communication between different devices.
- **Threshold and Ceiling:** *"The "threshold" is how difficult it is to learn how to use the system, and the "ceiling" is how much can be done using the system. The most successful current systems seem to be either low-threshold and low-ceiling, or high threshold and high ceiling. However, it remains an important challenge to find ways to achieve the highly desirable outcome of systems with both a low threshold and a high ceiling at the same time"*. A deep-rooted challenge for software tools is to address the dichotomy between the learning curve and the power of the system, that is the complexity of the tasks that users can accomplish with the system. For instance, a very powerful tool (high-ceiling) may not have been adopted because of its high-ceiling. How to build tools with high-ceiling and low-threshold is therefore an important research question, especially for ubiquitous interaction, which demands for powerful tools that can manage the intrinsic complexity of the environment. By abstracting hardware device at an atomic level (sensors) the framework aims at providing an easy entry for developers. For instance, it provides the same interface and functionalities for the same class of sensors (e.g., accelerometer, gyroscope and the like) independently of their specific implementation. In this way developers are relieved of the burden to learn how to program different implementations of the same hardware, which in many cases requires knowledge of different operating systems and programming languages—this feature aims at supporting the rapid prototyping of heterogeneous interactive environments.
- **Path of Least Resistance:** *"Tools influence the kinds of user interfaces that can be created. Successful tools use this to their advantage, leading implementers towards doing the right things, and away from doing the wrong things"*. Myers et al. [2000] and Greenberg [2007] affirmed that toolkits affect the way we find solutions to problems; it is therefore important that such software tools guide the developers or designers towards *doing the right thing* avoiding them to take a wrong path. The structure and organization of the code have to reflect in the kind

of interfaces that can be created, having developers the feeling that they succeed in creating exactly what they wanted in a seamless way. The framework supports the development of multi-device environment by providing an infrastructure for uniform inter-device communication. In this case, developers should be able to create proper interaction for different kind of input devices without changing the underlying infrastructure.

- **Predictability:** *"Tools which use automatic techniques that are sometimes unpredictable have been poorly received by programmers".* It is important, in the case of software libraries, that the exposed APIs are clear to the developers, who feel confident on the results of using particular methods, types or structures. Research efforts in this direction focused on the evaluation of APIs usability using, for instance, the cognitive dimensions framework [Green et al., 1996]. The objective of cognitive dimensions is to evaluate, by means of twelve factors, the way developers expect an API to work and what an API actually offers. Related to the role of predictability for software toolkits are the *Role expressiveness* and *Consistency* factors:

- *Role expressiveness* measures "how apparent the relationship is between each component exposed by an API and the program as a whole"
- *Consistency* deals with "how much of the rest of an API can be inferred once part of it is learned".

Clarke [2004] suggested that, in order to evaluate API usability, a scenario-based approach can be used. The same approach has been employed in the user test presented in Section 6.3. This kind of approach requires the user to accomplish some programming tasks, in a specified context, using the API. After each task, or at the end of the experiment, user are asked to answer some questions related to each one of the cognitive dimensions: in this way it is possible to review the API and find failure or success patterns that can guide to the redesign of the API itself. Clarke [2004] provides some sample questions for the *role expressiveness* category, which are: (a) "When reading code that uses the API, is it easy to tell what each section of code does? Why?", (b) "Are there some parts that are particularly difficult to interpret? Which ones?" and, (c) "When using the API, is it easy to know what classes and methods of the API to use when writing code?".

- **Moving Targets:** *"It is difficult to build tools without having significant experience with, and understanding of, the tasks they support. However, the rapid development of new interface technology, and new interface techniques, can make it difficult for tools to keep pace. By the time a new user interface implementation task is understood well enough to produce good tools, the task may have become less important, or even obsolete".* This is another important issue to take into

account for a software tool that targets the development of ubiquitous systems. Experimentation with new input devices, modalities and techniques has increased in the last years and therefore tools are needed that both support the requirements of ubiquitous environments and provide a certain degree of longevity, for example by demonstrating their modularity or extensibility. The proposed framework, through primitive data types and device abstraction, aims at providing a platform to model future device and interactions.

6.2 Case study: digitally-augmented product shelf

The framework has been used to develop the case study of a digitally-augmented product shelf. The idea was to convert any physical surface and object into interactive surfaces that merges digital data with the real space. This system enables customers to conduct searches in store, learn about product features and talk to a virtual expert to get advices. The product shelf is based on the TESIS technology (see the description in Section 4.1.3), that can transform any standard shelf into an interactive surface, enabling customers to experience the combination of live product interactions enriched with the vast amount of information available on the web in an engaging and interactive way, exploiting augmented reality technology. Figure 6.1 shows how the development of the use case followed the framework reference architecture. TESIS has been modeled as a *TObject*, while the user's fingers and the objects of the product shelf as *PObjects*. The framework APIs allows to get two dimensional data from RGB and depth cameras (*Hardware Abstraction* layer). The original touch recognition algorithm from TESIS has been implemented as a software middleware for the framework, exploiting the hardware abstraction mechanism and the common data model to enable touch input (*Input Interpretation* layer). Then, touch-responsive widgets has been developed using the functionalities of *Application* layer. The digital user interface is projected on the touch-enabled physical surface by means of a pico-projector. Figure 6.2 shows the software interface of the augmented shelf. It provides the following functionalities to customers:

- Get detailed information on a specific product.
- Compare pricing.
- Access the Web to read unbiased reviews.
- Watch video presentations of a specific product.
- Call for assistance.

Main interaction is touch-based: customers touch on a specific virtual button displayed on the surface to activate desired content. The Microsoft Kinect has been used to recognize

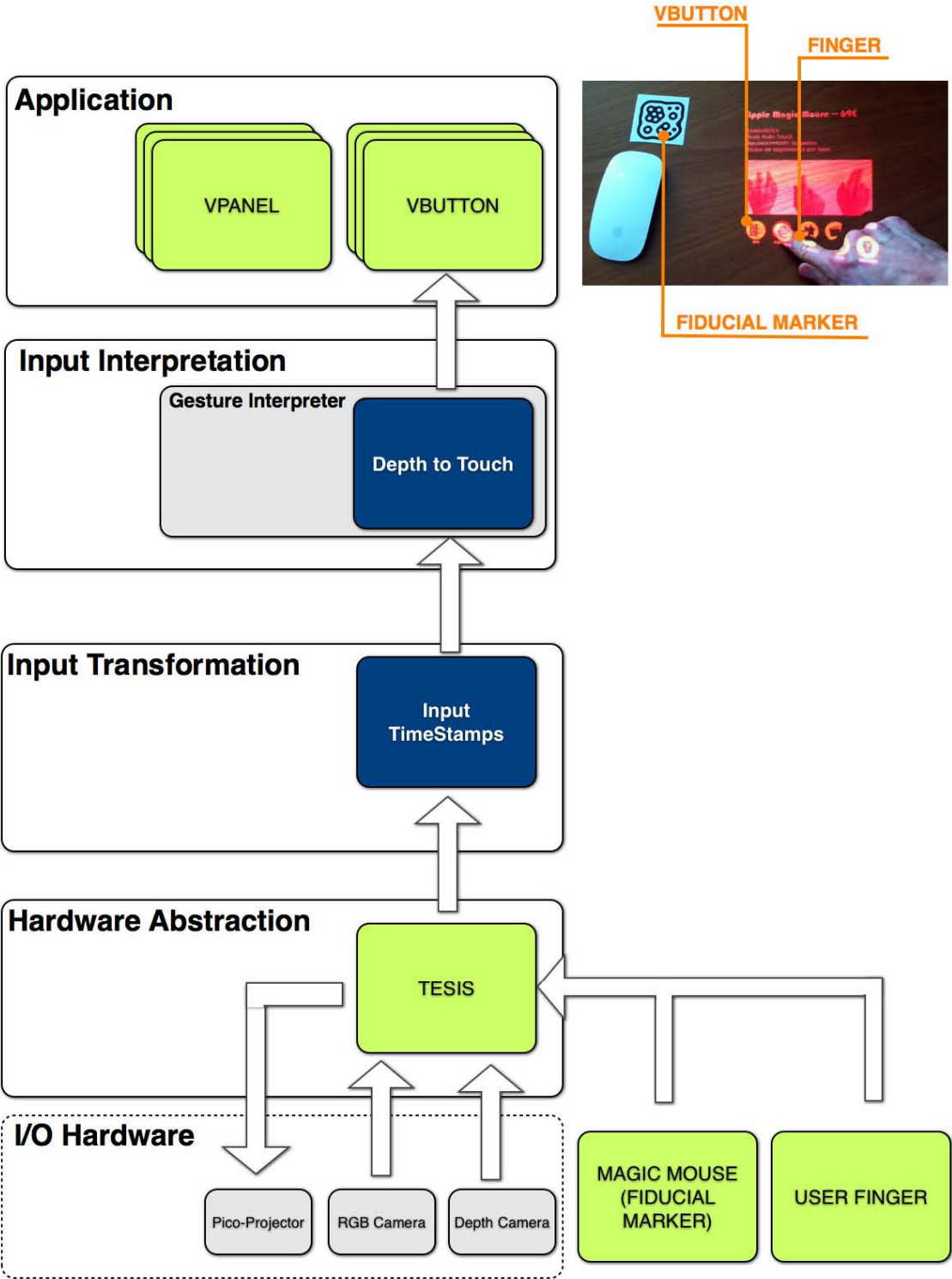


Figure 6.1: The digitally-augmented product shelf use case in the framework architecture.



Figure 6.2: Touch-enabled User Interface.

touches by means of depth imaging, thus allowing users to interact with digital elements blended with the real world in the same interactive space. Physical objects (the products) are recognized by means of visual markers, such as the code used in the reacTIVision library [Kaltenbrunner and Bencina, 2007]. The same visual codes are also exploited for user identification: for instance a user can display the code in his mobile phone which is scanned by the Kinect RGB camera, thus allowing to recognize the user identity. In this way the system can provide personalized information, such as recommendation of related articles, gathered directly from the Web. The system has been implemented using the Java APIs of the framework and the Processing IDE.

6.2.1 Goals

This use case has been exploited to answer the following questions with respect to three of the five categories from Myers et al. [2000]:

- **The parts of the user interface that are addressed:** does the framework succeed in implementing the application? which kind of real applications can be developed with the framework?

- **Path of Least Resistance:** is it possible to create proper interaction for different kind of input devices without changing the underlying infrastructure?
- **Moving Targets:** how does changing the input device affects the development?

Threshold and Ceiling and *Predictability* are addressed in the user evaluation, reported in the next Section 6.3, in which further insights are also provided for the *Moving Targets* dimensions.

6.2.2 Discussion

Findings from the development of the use case with respect to the three dimension to test are reported here.

- **The parts of the user interface that are addressed:** The use case demonstrated that is perfectly feasible to build a real application that makes use of heterogeneous devices and interaction techniques using the framework.
- **Path of Least Resistance:** By providing a comprehensive infrastructure that abstracts from devices, it has been possible to program the whole system without changing the programming rationale, which also demonstrates the low viscosity of the APIs. The low viscosity is one of the requirements, as described in Section 4.2.3.5. Moreover, the definition of a common model for the data also helps to follow specific strategies in the development, avoiding to take wrong paths.
- **Moving Targets:** Changing the input device did not require to install any third party library or start working with the rationale of a different APIs within the same project. The framework integrates both the touch input and the recognition of visual markers, thus facilitating the development of an integrated solution. Moreover, custom devices can be defined by means of XML configuration file. In this case, for instance, depth data for touches and RGB data for the visual markers have been retrieved from the Kinect. The *TESIS TObject* also features a pico-projector for visualizing the interface on a physical surface. If in future the system would be enhanced, for instance, with motion sensing capabilities, data from sensors such as accelerometers embedded in smartphones can be easily captured by the framework and integrated in the same scenario. Again, there would be no need to look to an external solution. As the user study demonstrates, the consistency of a comprehensive architecture saves development time together if compared with the combination of different, unrelated libraries.

6.3 User test

In the context of ubiquitous interaction, the trend is to extend the number of different input devices the user is capable to interact with; they can be either traditional ones such as mouse and keyboard, or exploit new types of sensors like tactile surfaces, accelerometers, RGB and infrared cameras. The proposed framework serves as an abstraction layer for different devices, facilitating the rapid prototyping of interfaces that use the aforementioned hardware. The goal was to develop a richer interaction experience, in a fast and simple way, by integrating different input device within a comprehensive architecture. This experiment aimed at collecting parametric data produced during the programming of a prototype (e.g., task completion time) as well as quantitative and qualitative data to determine the veracity of the hypothesis below. In particular, the objective was to verify performances of the software framework in situations where it is necessary to program the behavior of different interaction devices in the same scenario. To this end, a scenario-based approach [Carroll, 2000] has been used, as suggested by Clarke [2004] for the evaluation of API usability. To carry out this experiment, subjects with a unique profile specified later in this document have been selected. These individuals must build a physical/digital prototype using currently available software libraries or the proposed framework. In the rest of this section, the scenario package is described, including: (a) the goals of the experiment and the metrics for the evaluation, (b) participants, (c) an overview of the scenario, tasks breakdown and supporting material such as pre- and post-test questionnaires.

6.3.1 Goals

The experiment has been designed to test whether the framework facilitates the implementation of interaction mechanisms with heterogeneous devices using a specific programming environment. These concepts can be expressed through the following null hypothesis:

H_{n1} : The framework does not affect positively the efficiency (development time) in the development of prototypes that make an integrated use of different devices for ubiquitous interaction.

The alternative hypothesis is:

H_{a1} : The framework positively affects the efficiency (development time) in the development of prototypes that make an integrated use of different devices for ubiquitous interaction.

6.3.2 Participants

16 subjects (15 male and 1 female) performed the experiment: 5 undergraduate students, 10 post-graduate students and 1 researcher of the Computer Engineering Department at Universidad Carlos III de Madrid. Graduate students are from the Computer Science degree. Post-graduate students are enrolled in a Master course in Computer Science (5 subjects), focusing on HCI research, or enrolled in a Ph.D. program in HCI (5 subjects). In terms of target population, due to their technical background and skills, the subjects can be considered as developers who want to use the framework for the programming of ubiquitous environments. Information on the profile of the subjects will be gathered through a pre-test questionnaire (in Appendix C).

6.3.3 Design

For the evaluation of the hypothesis, two tasks have been designed. These tasks consist in developing a proof of concept that makes use of an accelerometer, a touch sensor and buttons using the framework and without using it, but instead specific APIs. The rationale behind this choice is to stress on the benefit of using a unified approach when programming interactive behaviors of different devices. In the tasks, the user is asked to program the behavior of an accelerometer and buttons (real or virtual) embedded in two existing devices (Nintendo Wii Remote Controller and Apple iPad) using three different software libraries (wrj4P5¹, oscP5² and the proposed framework). The programming environment used for the tasks is the Processing platform. wrj4P5 and oscP5 was the only available choices to program the interaction with Wiimote and iPad in the Processing environment. The experiment follows a $2 \times 3 \times 3$ fractional factorial design [Jonathan Lazar, 2010] with the following variables:

- **Independent variables (these are the three factors)**

- *Software Technology*, with three levels: wrj4P5 library, oscP5 library or the proposed framework.
- *Hardware Technology*, with two levels: Wiimote or iPad.
- *Interaction Component*, with three levels: real button (the physical button of the Wiimote), virtual button (a button widget on the iPad) or accelerometer.

- **Dependent variable**

- *Task completion time.*

¹<http://wrj4p5.sourceforge.jp/>

²<http://www.sojamo.de/libraries/oscP5/>

In a factorial design, not all the elements combination might be valid and, in this case, two of the six combinations resulting from software and hardware technologies (Table 6.1) do not exist because the oscP5 is used only for the iPad and wrj4P5 only for the Wiimote.

	Framework	wrj4P5	oscP5
Wiimote	F_w	L_w	–
iPad	F_i	–	L_i

Table 6.1: Combinations of the two factors *Software Technology* and *Hardware Technology*.

F_w : Framework with Wiimote, F_i : Framework with iPad, L_w : Library (wrj4P5) with Wiimote, L_i : Library (oscP5) with iPad.

Therefore, the original 2x3 part of the design has been condensed in a 2x2 design, resulting in the 4 levels presented in Table 6.1, which are F_w , F_i , L_w and L_i . In this experiment, the objective was not to test the way different components affect user interaction, a physical button in the Wiimote or a virtual button activated through a touch screen. Rather, it focuses on the performances of the framework for the development of interactive appliances. Therefore, the implementation of the button for the two hardware platforms (Wiimote and iPad) has been abstracted to its logical function: this allowed to merge the two levels *real button* and *virtual button* of the *Interaction Component* factor into one single generic level *button*, independently of the implementation. This design decision has been taken because it would not been possible to test the two levels for the different platforms, since the Wiimote does not offer a touch screen to implement button widgets. Moreover, in this way I could highlight the behavior of a comprehensive solution that abstracts hardware implementations through the notion of primitives data types and make a direct comparison with two separate libraries used in the same project. Due to this simplification, rather than the 12 runs that would have been required for the 4x3 part of the experiment (4 levels from the *Software plus Hardware Technology* factor and 3 from the *Interactive Component* factor), this experiment required only 8 (4x2) runs, as shown in Table 6.2. A *within subject* design

	F_w	F_i	L_w	L_i
Button (T1)	A	B	C	D
Accelerometer (T2)	E	F	G	H

Table 6.2: The final runs of the experiments: combination of *Software and Hardware Technology* with *Interaction Component*. The letters represent each run of a 8x8 Latin Square. The development of the button was tested in Task 1 (T1) and the accelerometer in Task 2 (T2).

has been used for this experiment for all the factors. According to MacKenzie [2002] there are at least two reasons to prefer a within subject design when possible: "*first*,

fewer participants are needed in a within-subjects design since each participant is tested on all levels of a factor. Although more testing is required for each participant, there is an advantage in having fewer participants overall, since recruiting, scheduling, briefing, demonstrating, practicing, and so on, are easier if there are fewer participants". In this experiment there is no *interference effect*, which is experimented when conflicting skills to operate one or another device are involved. In this case the skill required to operate one device tends to inhibit, block, or otherwise interfere with, the skill required for the other device. In this experiment the skills to operate the Wiimote or the iPad do not affect the development. The use of the software APIs for each library could interfere but this effect is mitigated in the experiment by the training that was given to all testing subjects. During the training they learnt the basics of the Processing IDE, accelerometers and the expected result of the task. In addition they were also given fifteen minutes to explore the APIs of the library being used during the experiment. The order of the tasks was balanced to account for any *learning effects*. Participants were randomly assigned to conditions for each task, following a 8x8 balanced *Latin Square*³ [Cochran and Cox, 1950], given the fact that a full-random approach would have required the impractical amount of $8! = 40320$ users. The 16 participants have been divided into eight groups of two, and each group has been assigned to one of the rows in the Latin Square as shown in Table 6.3. Compensating the bias introduced by the *order of presentation* of the tasks [MacKenzie, 2002] also allow to test the learnability of the framework API, under the *threshold and ceiling* dimension, independently of task-specific conditions. Large deviations due to the learning effect or *asymmetric skill transfer* [Poulton and Freeman, 1966] are not expected because of the great differences between the three libraries.

Group 1	A	B	C	D	E	F	G	H
Group 2	B	A	D	C	F	E	H	G
Group 3	C	D	A	B	G	H	E	F
Group 4	D	C	B	A	H	G	F	E
Group 5	E	F	G	H	A	B	C	D
Group 6	F	E	H	G	B	A	D	C
Group 7	G	H	E	F	C	D	A	B
Group 8	H	G	F	E	D	C	B	A

Table 6.3: The 8x8 Latin Square for counterbalancing.

The particular design of the tasks allows to frame the hypothesis, in the case of the development time, in the following way: given A the time⁴ for Wiimote without the framework, B the time for iPad without the framework and C the time with the framework (in this case we assume that the hardware technology is non influential, because of the hardware abstraction), therefore $A + B \gg 2C$, with $A \cong C$ and $B \cong C$. We can assume

³<http://goo.gl/BcLbP>

⁴the time the subject needs to program the requested behavior

that, without any prior knowledge of the software libraries for the two devices and the framework, the programming time is more or less the same ($A \cong C$ and $B \cong C$) but using two different technologies ($A+B$) in the same project will increase the programming time if compared to the case of using one single library ($2C$). The experiment was organized in individual sessions in a controlled setting. Subjects used a Mac OS X machine in the DEI Laboratory at Universidad Carlos III de Madrid. The “*Think Aloud*” protocol was used and each session was supervised by an investigator who interacted with the subjects to introduce the experiment, the task and to answer questions or doubts during the session. However, in no case the investigator helped the subject in the development of the task. The same investigator interacted in each session, to avoid possible biases introduced by the change of experimenter (e.g., different language and non verbal communication channels). The investigator took note of the time during the task (e.g., beginning and end of the task) and subject’s comments. The sessions was audio and video recorded with the previous consent of the participants (see the consent form in Appendix C). The task and previous training lasted around 90 minutes [Nielsen, 2005]. The time was divided as follows:

1. five (5) minutes for a short introduction of the experiment,
2. five (5) minutes for the subject to fill out the pre-test questionnaire,
3. ten (10) minutes for the training,
4. fifteen (15) minutes for the subject to read the scenario and study the provided API,
5. sixty (60) minutes for the programming tasks (T1 and T2),
6. five (5) minutes for the post-test questionnaire.

Further explanations on the objectives of the evaluation and the overall research were given at the end of the evaluation.

6.3.3.1 Pre-test questionnaire and training.

In the first twenty minutes before the development tasks, the investigator asked the subject to fill the pre-test questionnaire (see Appendix C) and then the subject was given a brief tutorial on the usage of the Processing platform. The pre-test questionnaire has been designed to gather information on the background of the users, especially with respect to their programming skills and their knowledge of I/O devices for ubiquitous interaction. The tutorial consisted of the following steps:

- An introduction to the Processing platform: the language, the API, the concept of a Sketch, how to import libraries in Processing and how to use the graphical IDE.
- Practice: the subject programmed his first *"Hello World"* sketch. Also an introduction to what is an accelerometer and the underlying physics concepts was given.

6.3.3.2 Scenario and APIs

The scenario proposed to the subjects is reported in Appendix C. After reading the scenario, up to fifteen minutes has been dedicated to the study of the API of the provided libraries: the library to program in processing with an iPad (oscP5), the Wiimote (wrj4p5) and the framework. This step stopped when the subject thought he has gathered enough knowledge to start programming the prototype. In no case it can last more than fifteen minutes. The documentation provided for the API has been standard javadoc for the framework and oscP5 and a javadoc-like list of classes and methods for the wrj4P5 library (Figure 6.3). For the task we used a subset of the real APIs of the framework, which were only related to the tasks at hand. For this reason, in order to account for possible biases introduced by the different extensions of the API, participants were instructed on the classes to use in the case of wrj4P5 and oscP5. For instance, the wrj4P5 library allows to interface with several Nintendo Wii peripherals such as the Balance Board or the Guitar in addition to the Wiimote. However, participants were told that only the main class (Wrj4P5) to instantiate the connection with the Wiimote and the class for the Wiimote object (WiiRimokon) would have been needed in the experiment.

6.3.3.3 Tasks

In the programming tasks, participants were asked to program a software application that makes use of an accelerometer, touch screen (iPad) and buttons (Wiimote). The input sensors are used in the application to interact with a virtual 3D box. Two different cases was proposed to the subject for both devices (Wiimote and iPad): (1) using the framework that provides integration of devices within a unique solution and, (2) using the two different libraries in the same Processing sketch. As previously reported, the objective was to test whether an integrated solution affect positively the efficiency, in terms of time, in the development of prototypes that make use of different devices for ubiquitous interaction. A detailed description of the tasks proposed for the experiment is reported in Appendix C, as presented in the documentation given to the subjects. Before starting, participants were allowed to interact with the devices and see the results of the two tasks (T1 and T2).

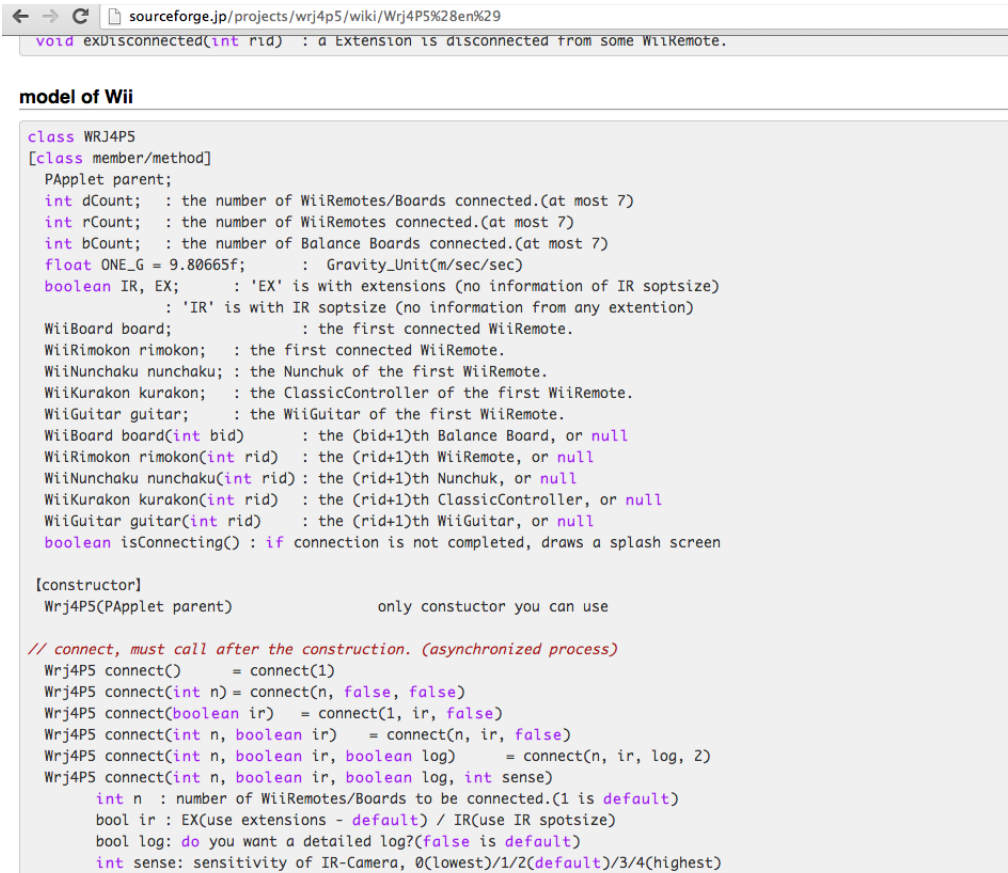


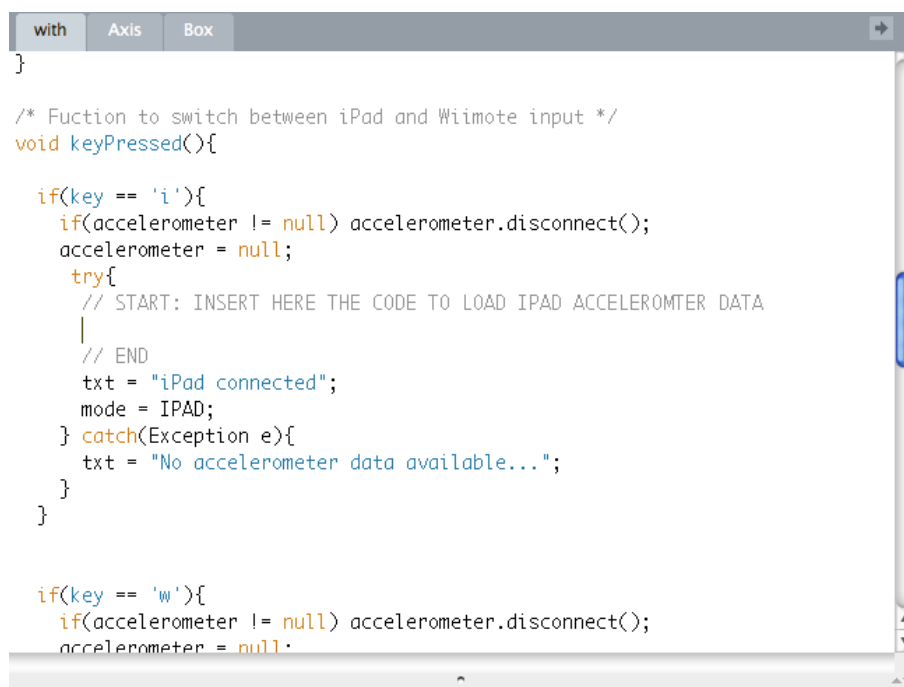
Figure 6.3: Excerpt of wrj4P5 API documentation.

- In T1 participants were asked to program the following: press the A button on the Wiimote or touch the screen of the iPad to change the color of a 3D box from red to green and,
- In T2 participants were asked to program the following: move the device (accelerometer data) to rotate the 3D box.

For the programming tasks the subjects did not have to write the entire code from scratch, which realistically would have taken too much time for an experimental session. A template was provided (see Figure 6.4), presenting the classes to use for the task. The template was designed to require the subject to type portions of code in specific places in order to implement, for each of the libraries:

- how to interface with the device,
- how to get relevant data from the appropriate sensor and,

- how to use the virtual or physical button data to change the color of cube or the acceleration data to rotate the cube.



```

}

/* Fuction to switch between iPad and Wiimote input */
void keyPressed(){

    if(key == 'i'){
        if(accelerometer != null) accelerometer.disconnect();
        accelerometer = null;
        try{
            // START: INSERT HERE THE CODE TO LOAD IPAD ACCELEROMTER DATA
            |
            // END
            txt = "iPad connected";
            mode = IPAD;
        } catch(Exception e){
            txt = "No accelerometer data available...";
        }
    }

    if(key == 'w'){
        if(accelerometer != null) accelerometer.disconnect();
        accelerometer = null;
    }
}

```

Figure 6.4: An example of the template provided to the subjects.

For example each library has a function to return acceleration data, but this value is not homogeneous between these functions, which can occur in possible errors when using different libraries in the same scenario. Again, each library proceeds in different ways in order to have an instance of a device object from which it is possible to access to the device functionalities. This affects the way subjects might behave during the development and therefore the time to complete the task. For example, the framework always instantiates an accelerometer object in the same way, independently of the underlying hardware and by using a configuration file. By contrary, the libraries for the Wiimote and the iPad implement different rationales to access to an instance of the hardware device.

6.3.3.4 Post-test and debriefing

Before ending each session, the subject was given a post-test questionnaire and then an informal talk took place, where the experimenter explained the real objective of the study and the subject could provide any idea, comment or suggestion. The informal chat was recorded as well. The post-test questionnaire (see Appendix C) was designed to include questions that make it possible to gather data according to the general usability of the proposed API and three of the five dimensions of the conceptual scheme from Myers et al. [2000]: *Threshold and Ceiling*, *Predictability* and *Moving Targets*.

6.3.4 Results and Discussion

In this section, the results of the experiment questionnaires (pre- and post-test) and the empirical analysis are presented. In the case of Likert [Likert, 1932] scale for responses to pre- and post-test questionnaires, histograms and box plots are employed to present data. Histograms give an overview of the distribution density of the samples. Box plots visualize the dispersion and skewness of samples. In particular, box plots have been made following the approach proposed by Montgomery [2008]: the middle bar in the box is the median. the lower quartile is the 25% percentile and the upper quartile is the 75% percentile. The upper tail is the highest value of the sample and the lower tail is the lowest value. Tails represent the theoretical bound within which it is likely to find all data points if the distribution is normal.

6.3.4.1 Pre-test Questionnaire

With the pre-test questionnaire, information on the participant profile has been gathered. Question 1 (Q1, "Of the following programming language and technologies, check those that you have personally used and are familiar with", in Figure 6.5) assures that participants all know the Java programming language (the framework implementation being evaluated is written in Java).

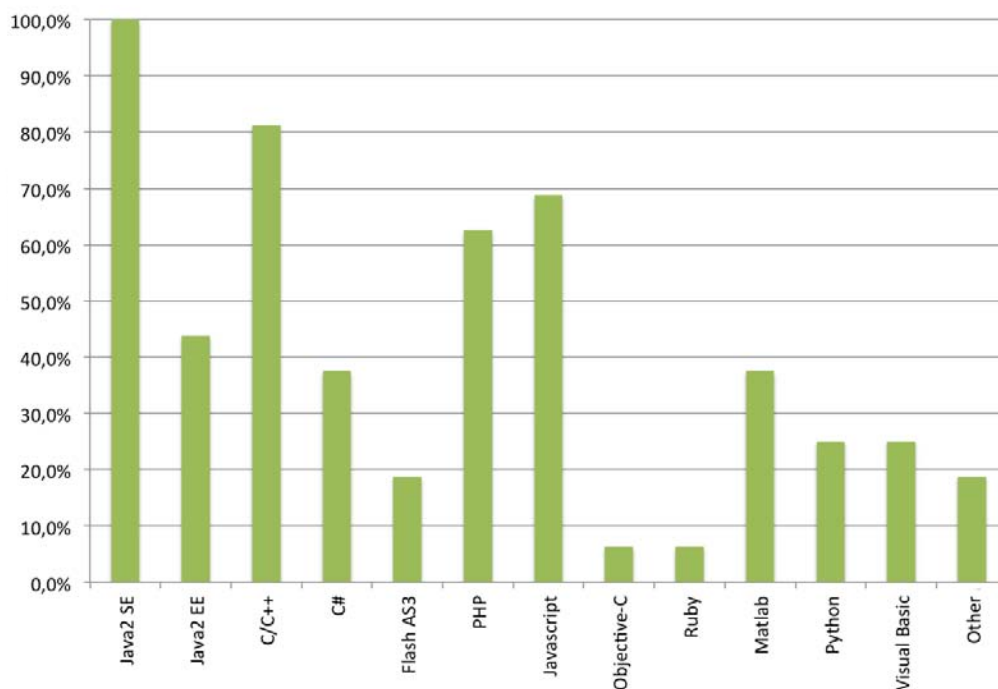


Figure 6.5: Pre-test questionnaire, Q1: "Of the following programming language and technologies, check those that you have personally used and are familiar with".

Q2, "Please rate your technical programming knowledge (programming paradigms, data structures, frameworks, etc.)", and Q3, "If you know/use the Java programming language, what is your level of proficiency with the language?", define the technical programming knowledge of participants. As shown in Figure 6.6, in this evaluation only participants with high programming skills have been selected.

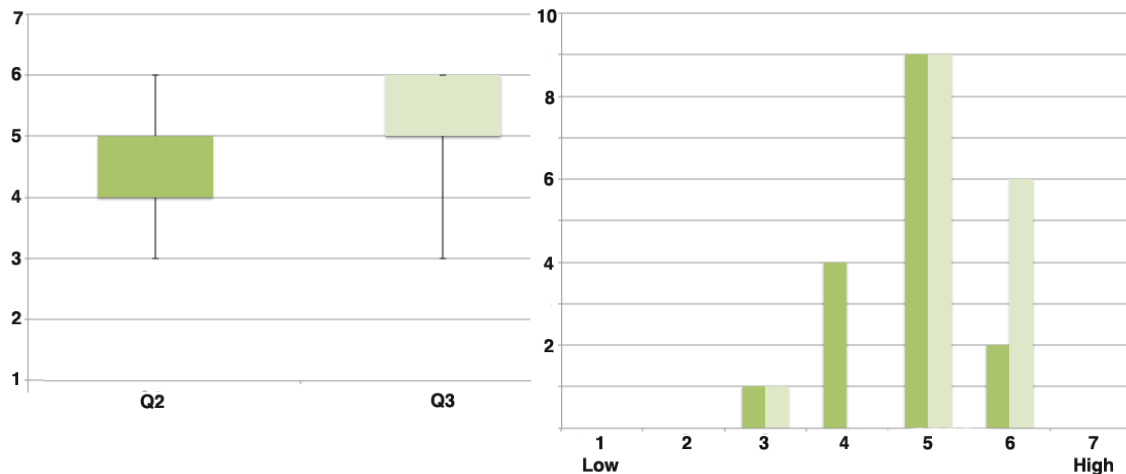


Figure 6.6: Technical programming skills of the sample population. Q2, in dark green: "Please rate your technical programming knowledge (programming paradigms, data structures, frameworks, etc.)". Q3 in light green: "If you know/use the Java programming language, what is your level of proficiency with the language?". Box-plot on the left and distribution of the answers on the right.

63% of the participants have never programmed applications using interactive devices in general (Q8, Figure 6.7), while 62% did have an academic or professional background related to interaction design, human-computer interaction or ubiquitous computing (Q9, Figure 6.8)

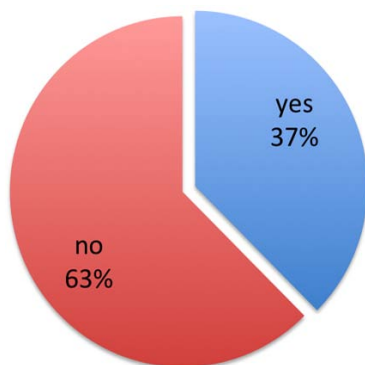


Figure 6.7: Q8: "Have you ever programmed interactive systems that make use of the aforementioned (Q7) hardware?".

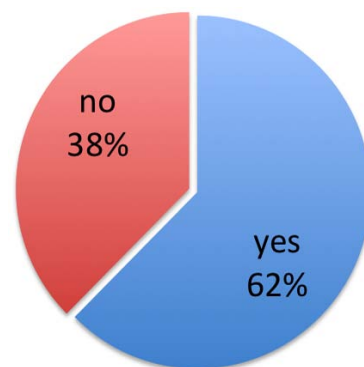


Figure 6.8: Q9: "Is your academic/professional background related to interaction design, human-computer interaction or ubiquitous computing?".

6.3.4.2 Post-test Questionnaire

All the participants completed the tasks. The post-test questionnaire is divided into four categories: *Overall*, *Threshold and Ceiling*, *Predictability* and *Moving Targets*. Questions are Likert-type scale (7 values, from 1 minimum to 7 maximum) and, for *Threshold and Ceiling*, *Predictability* and *Moving Targets*, there is a open question at the end of each section, to collect any personal comment of the participants. Question 1, "How would you define your experience of programming the interaction of the prototype *WITH* the framework?" (Q1: $\mu_{Q1}=6$, $\sigma_{Q1}=0.26$) and Question 2, "How would you define your experience of programming the interaction of the prototype *WITHOUT* the framework (with wrj4P5 and oscP5)?" (Q2: $\mu_{Q2}=3.20$, $\sigma_{Q2}=0.32$), summarized in Figure 6.9 and Figure 6.10, give a comparison of the overall rating of the framework and the two libraries (wrj4P5 and oscP5) with respect to the two tasks (T1 and T2, reported in 6.3.3.3). Analyzing the mean score for Q1 and Q2, the participant general perception of the framework is consistently better, compared with the combination of the two libraries, in all the single subquestions. One of the participants summarized her experience at the

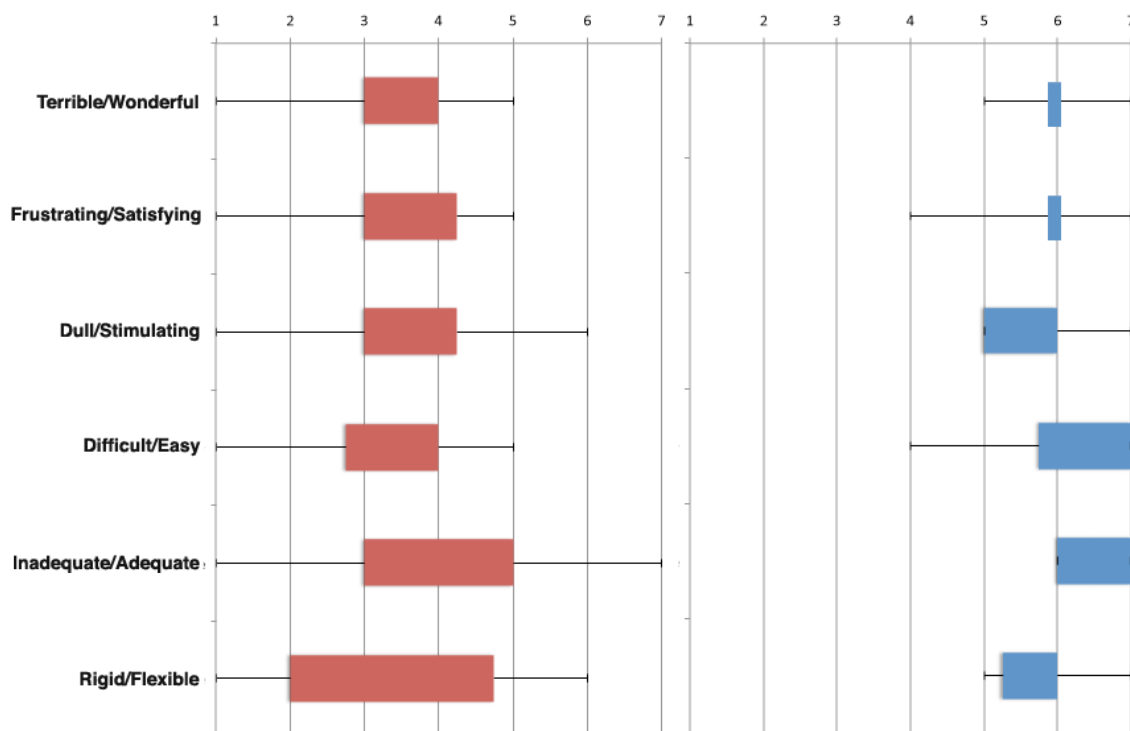


Figure 6.9: Box plot of the overall rating for the two libraries (on the left and red, Q2) and for the framework (on the right and blue, Q1).

end of the evaluation saying:

It was a nightmare programming with two different libraries in the same

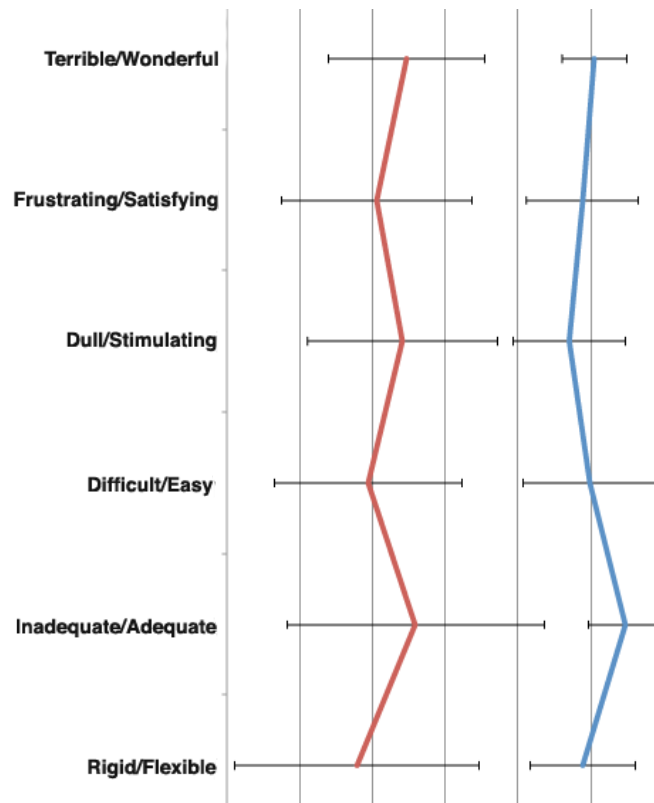


Figure 6.10: Direct comparison of the overall rating for the two libraries (red, Q2) and the framework (blue, Q1).

project! With the framework, on the other side, even if I found difficult to get used to it at the beginning, then it made the development straight-forward.

According to Q3, "When reading code that uses framework's Application Programming Interface (API), was it easy to tell what each section of code does? Why?", participants found easy to understand the API provided by the framework. For instance one of the participants reported that:

The documentation is clear and follows java standards.

Another one accounted that:

The names of properties and methods was intuitive enough so as to know what they do.

Q4, "How did the technical environment (Processing) make the programming tasks?" ($\mu_{Q4}=5.05$, $\sigma_{Q4}=1.29$) concerned the influence of the programming environment (Processing) on the tasks: according to responses (Figure 6.11) it did not have negatively influenced the experiment.

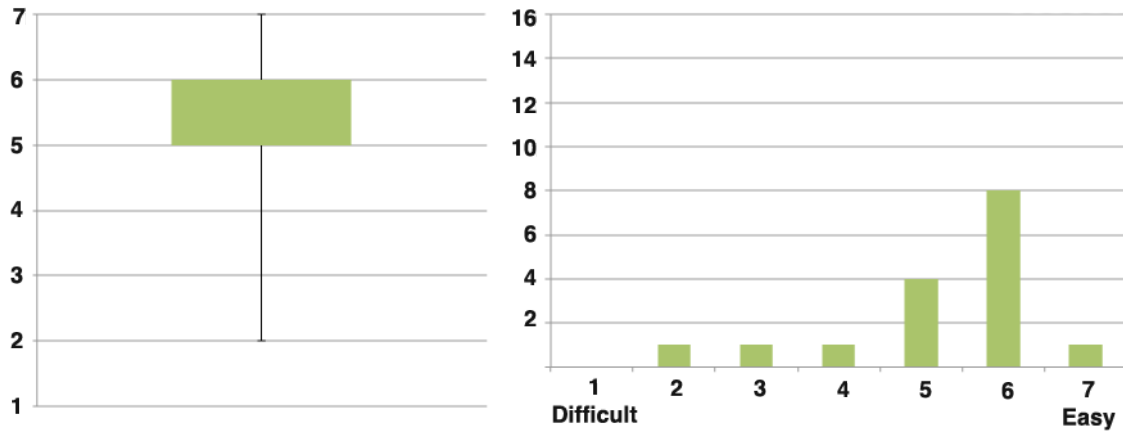


Figure 6.11: Q4: "How did the technical environment (Processing) make the programming tasks?" Box plot on the left and histogram on the right.

Participants agreed to use the framework for the development of ubiquitous systems, as shown by Q5, "Would you use the framework to develop interactive systems for ubiquitous environments?" ($\mu_{Q5}=6.18$, $\sigma_{Q5}=0.70$), summarized in Figure 6.12. Q1 and Q5 indeed show that participants have been overall satisfied with the framework in the programming tasks.

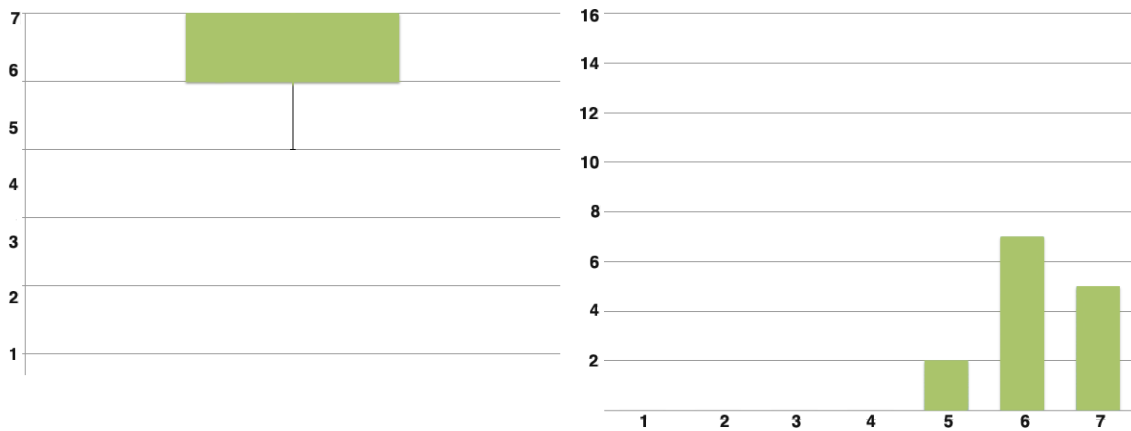


Figure 6.12: Q5: "Would you use the framework to develop interactive systems for ubiquitous environments?" Box plot on the left and histogram on the right.

Next, the discussion of the three factors — *Threshold and Ceiling*, *Predictability* and *Moving Target* — is reported, according to the items in the questionnaire used to extract the score for each factor.

Q6 to Q14 define the *Threshold and Ceiling* factor ($\mu_{th_and_ce}=5.80$, $\sigma_{th_and_ce}=0.72$), which is shown in Figure 6.13. Participants found easy to learn how to use the framework (Q6, "Learning how to use the framework was": $\mu_{Q6}=6.53$, $\sigma_{Q6}=0.63$). Analyzing Q12, "The ease of programming with the framework depends on the level of experience"

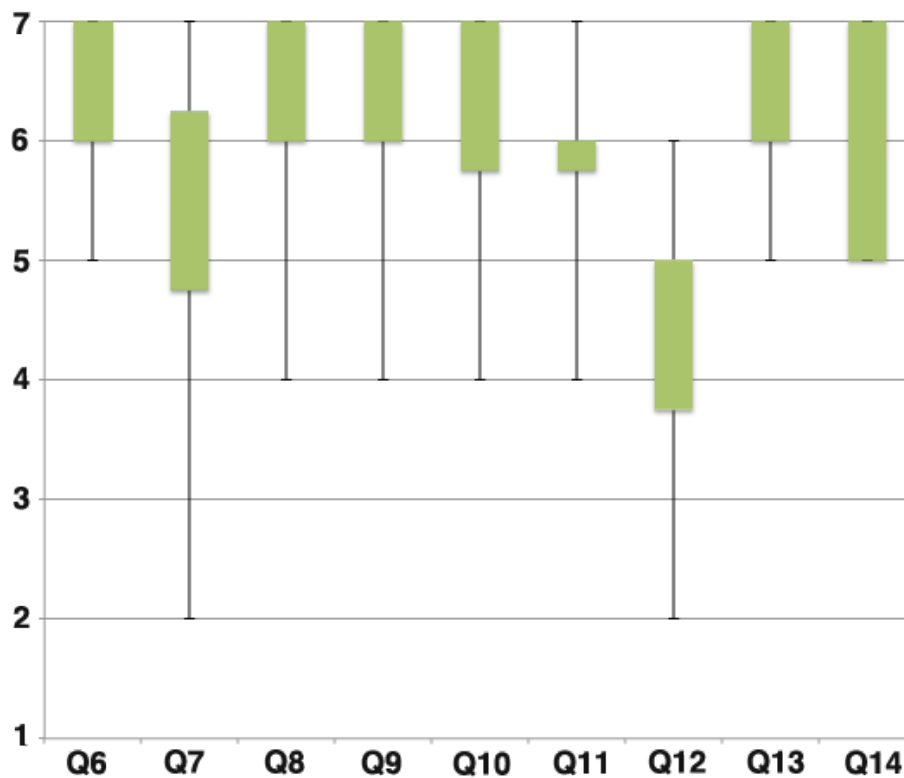


Figure 6.13: Threshold and Ceiling factor (Q6 to Q14).

($\mu_{Q12}=4.24$, $\sigma_{Q12}=1.31$), it appears that participants have divergent opinions on the level of programming experience required to use the framework, even if they all have high programming skills. The simplicity of the programming tasks did not allow to test to what extent the framework supports the development of complex interactions. Anyhow, according to open answers to Q16, "Please write your comments about how difficult is to learn how to use the framework and how much can be done using the framework here", it seems participants perceive the framework is powerful enough to support the programming needs of ubiquitous systems. They reported:

I think is quite easy to learn given the functionalities it provides. I think this framework would allow developers to program a wide range of applications for these kind of devices in a short time

and,

Apparently there are a lot of thing that can be done using the framework, which I could not have experimented in the evaluation.

and also,

Great! It is not necessary to learn how different APIs work in order to perform the same task with different devices.

Participants considered that they needed to write a fair amount of code to accomplish the tasks with the framework (Q15, *"How did the amount of code required for the framework for each subtask in this scenario seem to you?"*: $\mu_{Q15}=3.48$, $\sigma_{Q15}=0.73$), maybe slightly too little. Some participants, for example, complained that connection details are enclosed in the constructor of the class for the accelerometer or the button, while they would have preferred an explicit *connect()* method.

Q17 to Q21 define the *Predictability* factor ($\mu_{\text{predictability}}=5.69$, $\sigma_{\text{predictability}}=0.62$), which is shown in Figure 6.14.

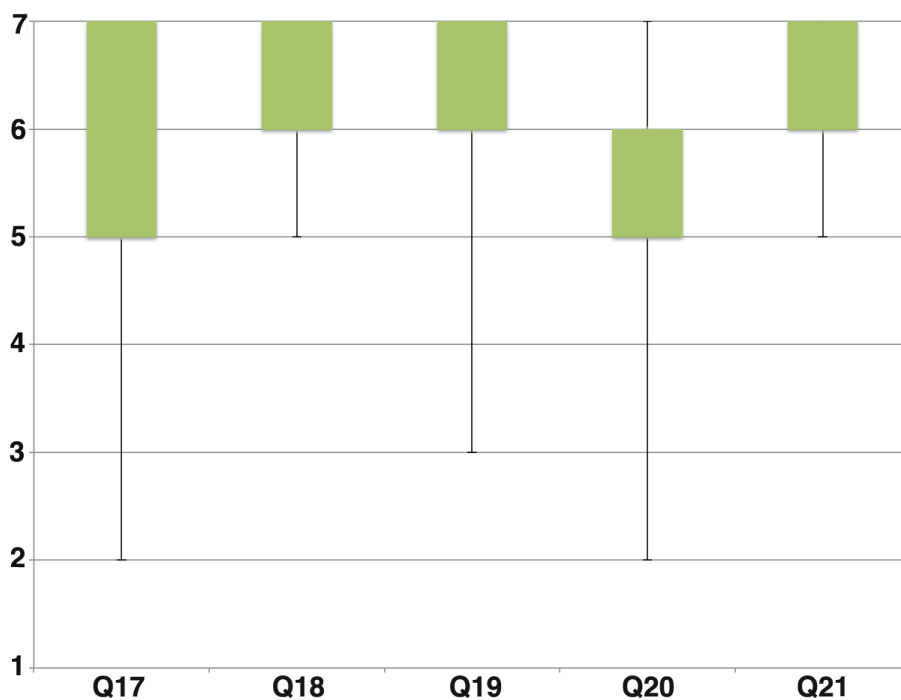


Figure 6.14: Predictability factor (Q17 to Q21).

Participants appreciated that the name of objects, methods and variables were clear (Q17, *"The names used for the framework's objects, methods and variables were"*: $\mu_{Q17}=5.40$, $\sigma_{Q17}=1.45$) and they found the framework to behave as expected (Q19, *"Did the APIs of the framework behave as expected?"*: $\mu_{Q19}=6.07$, $\sigma_{Q19}=1.11$). One of the participants reported that (Q22, *"Please write your comments about the predictability of the framework here"*):

[...] names of methods were clear, so it was easy to predict what each function does.

81.25% found the framework reliable (Q21, *"The framework is reliable"*: $\mu_{Q21}=6.20$, $\sigma_{Q21}=0.73$). 28.75% of the participants felt that the evaluation session did not give them enough experience with the framework to judge its reliability, thus they did not answered the question. Participants found the framework to be consistent (Q18, *"Were the names of objects, methods and variables of the framework consistent"*: $\mu_{Q18}=6.15$, $\sigma_{Q18}=0.66$), especially when changing from one task to the other. For instance, most of them complained that the wrj4P5 library does not treat accelerometer actualizations as events, while it does so with Wiimote buttons. On the contrary, oscP5 always receives messages that are processed according to the Java event handling system. This difference in managing data from different devices negatively affected the time to complete the tasks with the two libraries, as demonstrated in the empirical analysis in the next section. With respect to the framework, participant praised that the comprehensive rationale endorses consistency:

It was very easy to code the same behaviour into both of the devices. Only is needed to know which button of which device has been pressed. Once it is known, the behaviour is programmed in the same way (same code).

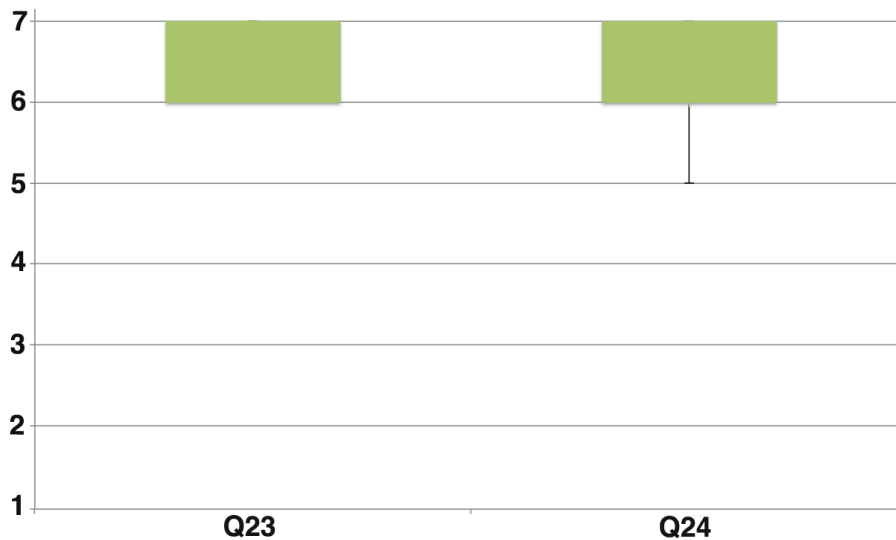


Figure 6.15: Moving Target factor (Q23 and Q24).

Q23, *"With the framework it was possible to create proper interaction for different kind of input devices without changing the underlying infrastructure"*, and Q24, *"Changing the input device (Wiimote and iPad) affected the development WITH the framework"*, define the *Moving Targets* factor ($\mu_{\text{moving_t}}=6.40$, $\sigma_{\text{moving_t}}=0.37$), which is shown in Figure 6.15.

Participants considered that the framework allows to program proper interaction without changing the underline infrastructure (Q23 : $\mu_{Q23}=6.67$, $\sigma_{Q23}=0.48$), because

of the hardware abstraction the XML configuration file. They also reported that changing the input device did not negatively affect the development with the framework (Q24: $\mu_{Q24}=6.14$, $\sigma_{Q24}=0.75$). On the contrary, it did negatively affect the development with the two different libraries, according to Q25, "*Changing the input device (Wiimote and iPad) affected the development WITHOUT the framework*" ($\mu_{Q25}=1.45$, $\sigma_{Q25}=0.63$). The comparison between Q24 and Q25 is shown in Figure 6.16. In this case the meaning of the Likert scale is inverted, meaning 1 "*Drastically*" and 7 meaning "*Not at all*".

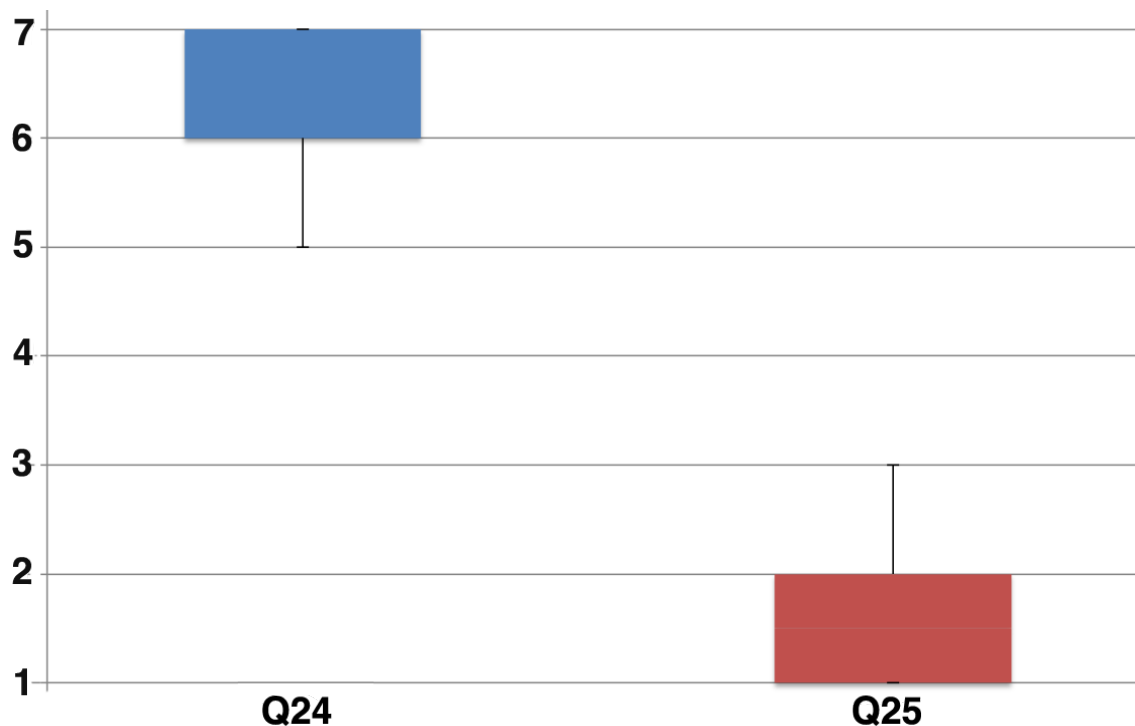


Figure 6.16: Comparison of Q24 and Q25 that shows how changing the input device (Wiimote and iPad) affected the development WITH (on the left and blue, Q24) and WITHOUT (on the right and red, Q25) the framework. 1 meaning "*Drastically*" and 7 meaning "*Not at all*".

Figure 6.17 shows the overall comparison of the three factors of the post-test questionnaire, plus the *Overall* factor that measures the overall acceptance of the framework (Q1 and Q5). From participants answers, the proposed framework seems to accomplish with the requisites defined by Myers et al. [2000] for successful toolkits for the three dimensions considered in this evaluation.

6.3.4.3 Empirical Analysis

For the empirical analysis, the time to complete the task has been considered. The primary result was that programmers were dramatically and significantly faster — between

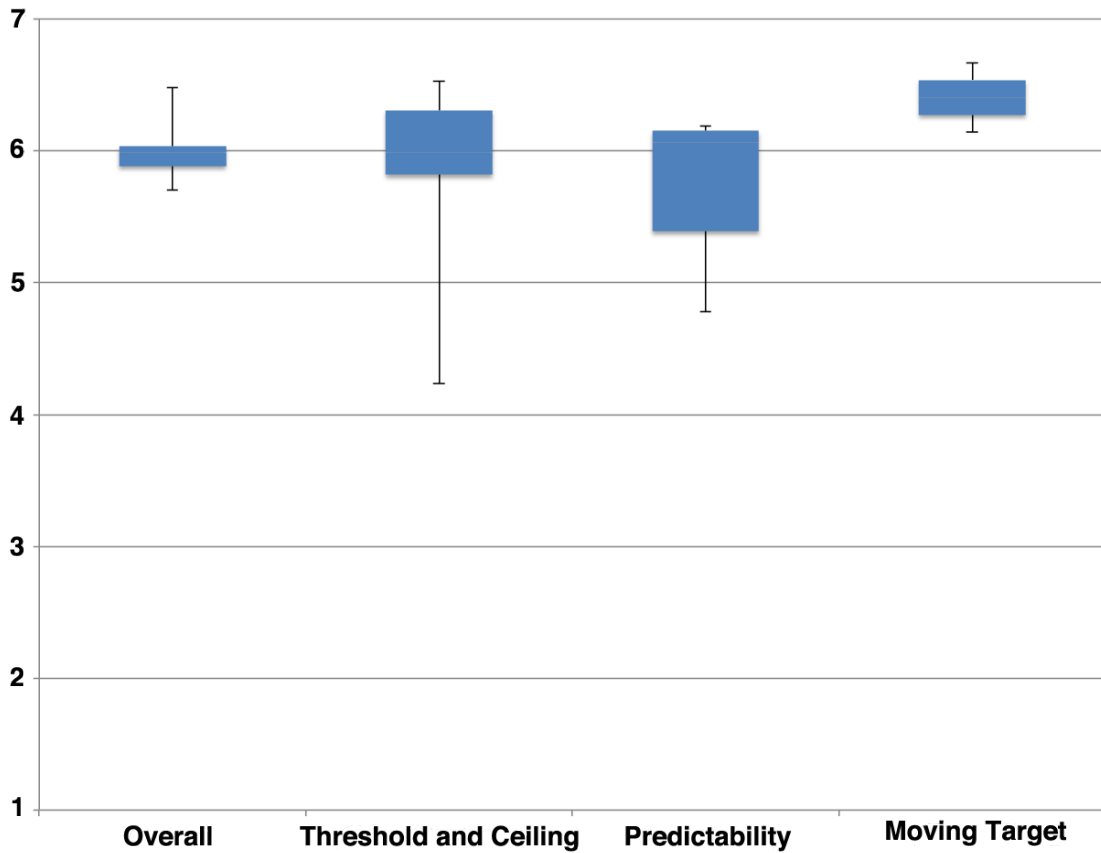


Figure 6.17: Summary of the post-test questionnaire results using the scores for the three factors defined by Myers et al. [2000]: *Threshold and Ceiling*, *Predictability* and *Moving Target*. A fourth factor, *Overall* measures the overall acceptance of the framework.

2.4 and 11.2 times faster — at using the framework than combining different libraries for the two tasks (2.75 mean value). See Figure 6.18 for total times for the two tasks both with or without the framework.

The total time to complete the two tasks with the framework has been $\mu_{fr}=12m\ 00s$, $\sigma_{fr}=5m\ 32s$. Using the two libraries participants accomplished the two tasks in $\mu_{lib}=33m\ 55s$, $\sigma_{lib}=8m\ 11s$. In the first task (T1, described at page 133), which consisted of changing the color of the 3D cube, participants spent an average of $\mu_{T1fr}=5m\ 55s$, $\tilde{x}_{T1fr}=5m\ 34s$ and $\sigma_{T1fr}=4m\ 35s$ programming the behavior of physical and virtual buttons with the framework, compared with the $\mu_{T1lib}=12m\ 16s$, $\tilde{x}_{T1lib}=14m\ 15s$ and $\sigma_{T1lib}=8m\ 56s$ with the two libraries. The second task (T2, at page 133), which asked to rotate the 3D cube, shows similar results: $\mu_{T2fr}=4m\ 55s$, $\tilde{x}_{T2fr}=3m\ 49s$ and $\sigma_{T2fr}=4m\ 02s$ compared with $\mu_{T2lib}=16m\ 45s$, $\tilde{x}_{T2lib}=18m\ 13s$ and $\sigma_{T2lib}=11m\ 06s$. All the participants completed the two tasks with the four combinations of software libraries (with or without the framework) and input device (Wiimote or iPad). In Figure 6.19, average time for each subtasks is reported, showing that the framework performed better

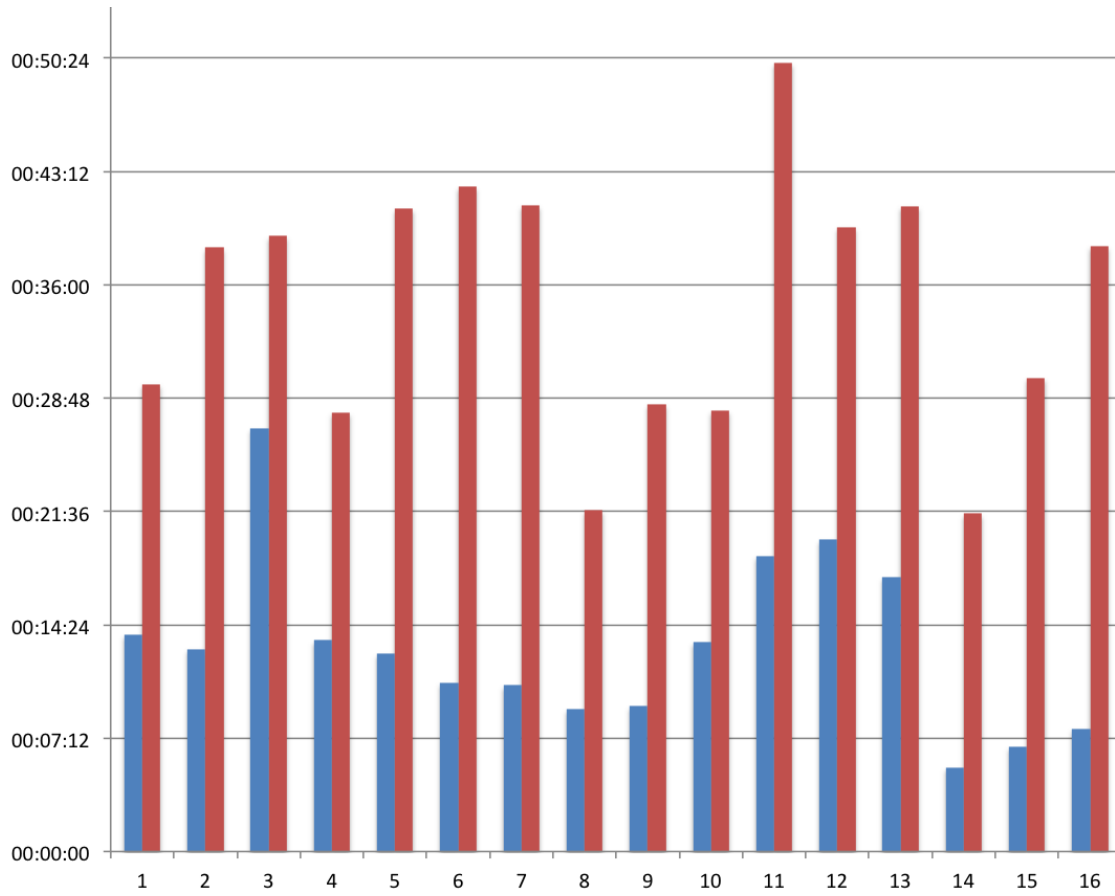


Figure 6.18: Total time for each participants to accomplish the two tasks WITH (in blue) or WITHOUT (in red) the framework. On the x-axis the participant and on the y-axis the time are plotted.

under each condition.

T1 and T2 with the framework have similar completion time; this suggests that the framework does succeed in providing an abstraction level that allows programmers to work independently of the underlying hardware device. This is also demonstrated by programmers strategies with the framework: it was observed that 12 out of 16 participants (75%) preferred to program the two devices in parallel rather than sequentially. Even if they were instructed to start with one of the two, once instantiated the object for the first device, they immediately instantiate a second one for the other because they realized that:

Just by changing the names and id in the constructor, one changes the input device.

On the other side, with the two libraries they always programmed sequentially starting and ending the task with one library before passing to the other one. Worst average completion

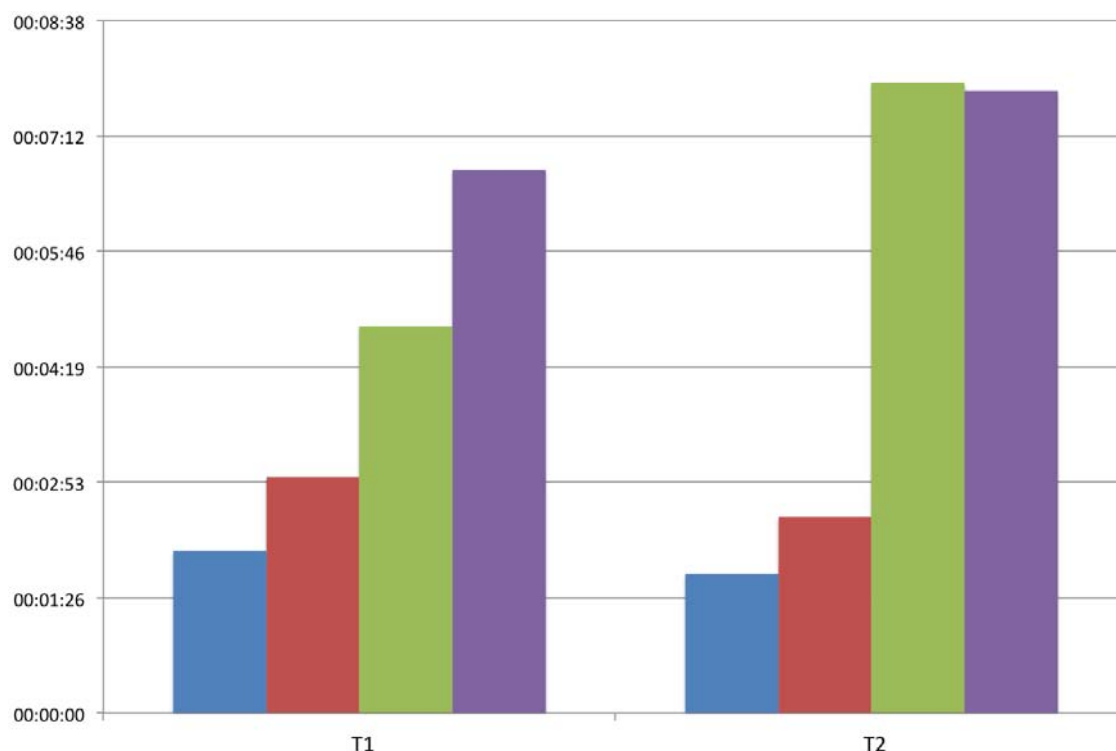


Figure 6.19: Average times for each task and condition. Blue: Wiimote w/ framework. Red: iPad w/ framework. Green: Wiimote w/o framework. Purple: iPad w/o framework.

time has been experimented in T2 both with the Wiimote and iPad. In the case of the Wiimote it can be explained by the fact that the poor and non-standard documentation of the API did not help users in understanding that accelerometer updates were stored in a class variable. Most of participants (13 out of 16, 81.25%) expected to have an event for the accelerometer, since they are used to handle event message according to the Java rationale. One user did not find this behavior particularly problematic, though he reported that the real problem was in the lack of consistency between libraries:

I don't mind if the accelerometer does not fire events. What I do care is that these sensors must be managed always in the same way even with different libraries. It puzzled me to have to change the logic between wrj4P5 and oscP5.

The problem with oscP5 could have been that users found it difficult to understand how OSC message are constructed and parsed. Most of the time spent for both T1 and T2 with oscP5 has been devoted in understanding how to recognize from which component the message came from, because of the poor name of methods and variables. It was difficult for them to find out that the variable *addrPattern* contained the id of the component and that there was a method that allowed for comparisons,

`boolean:checkAddrPattern(theAddrPattern:String)`. T2 had an additional challenge, if compared with T1, which justify the difference in time. In T1 participants had only to check if the particular widget button was pressed, which is accomplished by simply checking the `addrPattern` of the button. In T2, they also had to extract the actual acceleration on the y axis. To do so, they had to find the `OscArgument:get(theIndex:int)` method, which grant access to values in the OSC message. Locating this method in the API was not easy. A Shapiro-Wilk normality test [Shapiro and Wilk, 1965] on time series for the total time with or without the framework does not reject the hypothesis that the distribution is normal (there is no evidence of a no normality distribution). With the framework $w=0.93$, without the framework $w=0.92$. For a 0.01 confidence interval, the threshold value is $p\text{-value}=0.84$, for 0.05 $p\text{-value}=0.89$ and for 0.10 $p\text{-value}=0.91$.

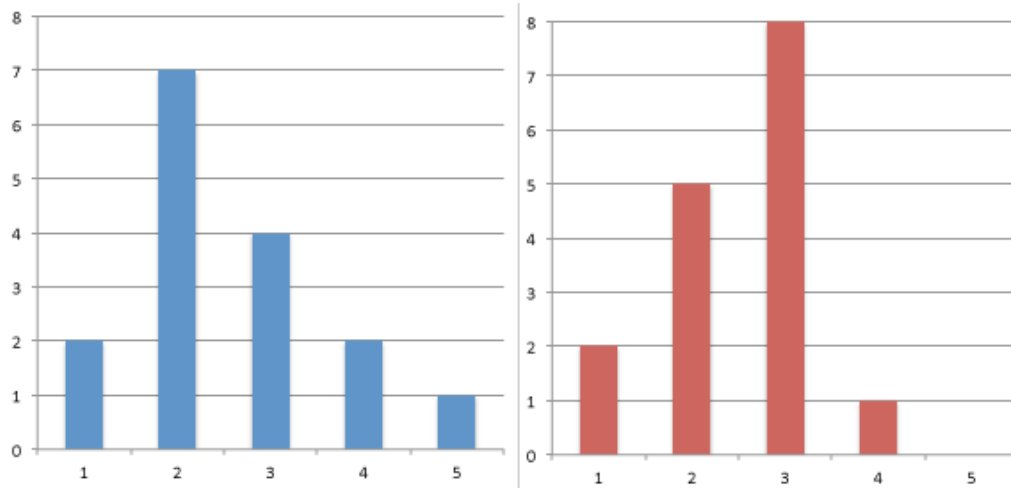


Figure 6.20: Frequency histogram of the two time series: WITH (in blue, on the left) and WITHOUT (in red) the framework.

Frequency histogram (Figure 6.20) and Q-Q plot normality test (Figure 6.21) highlight that the timing distributions are not normal and, therefore, a non-parametric test for statistical significance must be used — the Wilcoxon Rank Sum for Large Samples method was chosen [Wilcoxon and Wilcox, 1964]. Moreover, the timing data exhibited both *ceiling and floor effects* [Vogt and Johnson, 2011]. Ceiling effects arise when test problems are insufficiently challenging. Floor effects floor are just like ceiling effects but they are found at the opposite end of the performance scale. As Stylos and Myers [2008] suggested, in this case the Wilcoxon Rank Sum non-parametric test can be used. The Z score ($Z = 5.48$) is greater that the Z critical ($Z_{crit}=1.96$). Therefore it is possible to affirm, for a confidence interval of 95% that the null hypothesis concerning applications development time (H_{n1}) is rejected, thus demonstrating that the framework positively affects the programming of interaction when heterogeneous devices coexist in the same environment. The statistical power is 90% and the initial assumption on time is that the framework reduces the development time by a 40%, extracted from a pilot study with

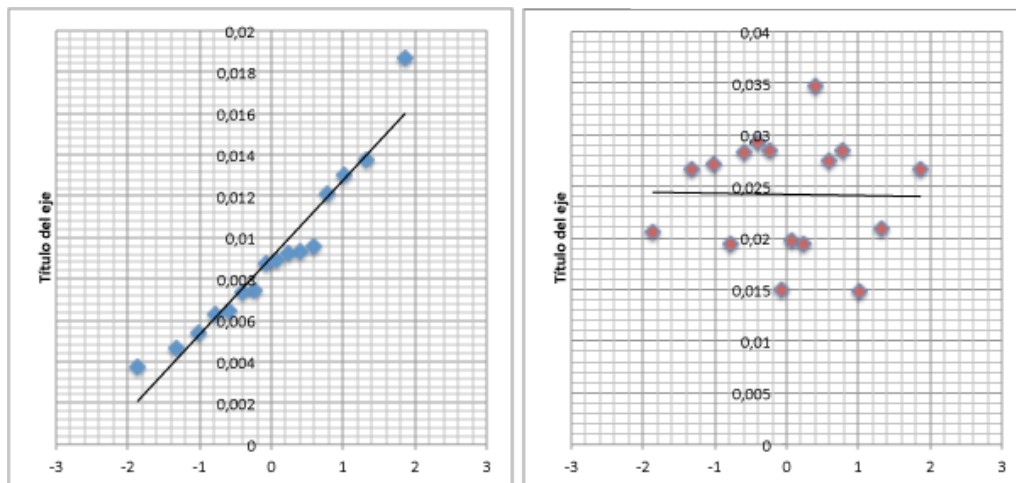


Figure 6.21: Q-Q plot for the two time series: WITH (in blue) and WITHOUT (in red) the framework.

12 participants. According to the formula for calculating the sample size from [Noether, 1987], 16 samples per group (with and without the framework) is enough to assure the statistical validity of the claim. Moreover, any statistically significant effect of task order or individual participant programming experience on task completion times has been found, showing that there has been sufficient counterbalancing.

6.3.4.4 Threats to validity

To evaluate the validity of the results, threats that may affect the experiment [Wohlin et al., 2012], according to Cook et al. [1979] are discussed.

Construct validity. It concerns the generalization of results to a theory and, in general, construct validity threats are mitigated by the experiment design. Indeed, *mono-operation* and *mono-method* biases have been avoided by the adoption of two tasks and four combination of software libraries and devices (Table 6.2). Mono-operation bias pertains to the independent variables: if a single version of a program is used in a single place at a single point in time, the full breadth of the concept of the program might not be captured. Construct validity can also be threatened when using a single method to measure a given construct (irrespective of whether the construct is acting as the dependent or independent variable). This is because the method used may introduce bias, changing the scores on the independent or dependent variable. This is known as mono-method bias. The *Inadequate preoperational explication of constructs* threat [?]p71]wohlin2012 — in other words, the construct is not well defined (operationally) — has been avoided by comparing the framework with the combination of the two other libraries in terms of the objective measure of task completion time.

Internal validity. In this case, it needs to make sure that there is a causal relation between the treatment and the outcome. All the subject were volunteers and, therefore, it can be expected that they are equally motivated, thus reducing the impact of users reaction with respect the passing of time during the experiment (*maturation* threat). The *history* threat has been addressed by performing the sixteen evaluations under the same circumstances, during normal working days, in a two weeks time span, from tuesday to thursday, from 10:30 to 18:30. The programming environment, Processing, does not affect negatively the experiment (*instrumentation* threat) as demonstrated by Q4 of the post-test questionnaire, which shows that the perception of the material provided (the Processing programming environment) was positive (Figure 6.11). The *instrumentation* validity is further assured by the fact that both pre-test and post-test questionnaires has been designed following the scheme proposed by Myers et al. [2000], questions proposed by Clarke [2004] to study API usability and also question from well-know usability and user experience questionnaires such as Purdue Usability Testing Questionnaire (PUTQ) [Lin et al., 1997] and Technology Acceptance Model (TAM) [Davis, 1985].

External validity. This aspect addresses the ability to generalize results from the experiment. The participants in our study were all Computer Engineering Ph.D., MsC, or BsC students (plus one HCI researcher). However because the programming and API exploration strategies manifested by the subjects matched those observed in previous studies with different participants [Clarke, 2004; Stylos and Myers, 2008], results can be generalized at least to programmers in general who exhibit *pragmatic* and *opportunistic* programming strategies[Stylos and Myers, 2008]. During the study programmers worked in a realistic manner to get accurate timing information and to gain insight into programmers' thoughts and assumptions while they worked. The think-aloud protocol was used, which likely affected their times. However, since the same protocol was used in both conditions, the relative time comparisons between the two conditions are still valid. In the experiment, participants used real versions of the two libraries (wrj4P5 and oscP5) and an ad-hoc version of the framework API. They therefore were free to navigate the internet to find examples, a common strategy when learning a new API [Stylos and Myers, 2008], or simply exploring the API without accessing any sample code. The three tasks in our study were smaller than most realistic programming tasks, so that we could test more tasks and to avoid extraneous task complications. Nevertheless, because of the similarities in work strategies we saw across our tasks, and because programmers often approach larger programming tasks by focusing on smaller subtasks, we feel that our results will generalize to different and larger tasks.

Conclusion validity. A statistical relationship between the treatment and the outcome is needed. Assumptions on the statistical methods used for the analysis have been respected (*violated assumptions of statistical tests* threat) and the test with the highest

power as possible as been chosen (*low statistical power* threat). Measures can be considered reliable (*reliability of measures* threat) since I took the time needed to complete the tasks under each level. However, also objective measures have been taken into account, which have been extracted from the questionnaires.

6.4 Summary

In this chapter the evaluation of the framework has been reported. It has been carried out following two different processes:

- **A use case.** The augmented product shelf use case allowed to demonstrate that the framework is powerful enough to actually model and implement a real scenario and it also allowed to exemplify how each layer of the architecture cooperates in the whole implementation. With respect to the dimension defined Myers et al. [2000] it also demonstrated that different input technologies can be integrated into a unique solution.
- **A user study.** The user study has been carried out with the objective to evaluate how the framework affect the development (in terms of programming time) when compared with state of the art libraries. The experiment has been designed to have subjects to program the behavior of different devices in the comprehensive environment provided by the framework or, instead, by integrating different libraries in the same project. The framework resulted to positively affect the programming of interaction (time reduced by at least a 40%) when heterogeneous devices coexist in the same environment, with a confidence interval of 95% and a statistical power of 90%.

7

Conclusions

Remember that death is not the end but only a transition.

—*Fatal tragedy*

DREAM THEATER, PROGRESSIVE METAL BAND

This research has provided evidences that **by encompassing heterogeneous devices into a unique design it is possible to reduce user efforts to develop for interactions in ubiquitous environments**. The research has been motivated by the need to ease the rapid prototyping of Ubiquitous Interaction, given that developers have to face several hindrances, such as software programming, hardware knowledge and the intrinsic heterogeneity of the environment where different devices and interaction techniques coexist.

The main goal of this work has been to define a conceptual model and a reference architecture that provide users with a comprehensive framework for managing device ecologies. The framework has been demonstrated to be able to:

- model interactive spaces where different devices coexist, as shown in Section 5.1 with two sample scenarios,
- implement real systems that make use of heterogeneous I/O devices and interaction modalities, as shown in Section 6.2 with the digitally-augmented product shelf case study and,
- reduce the time needed by developers to implement the interactive behavior of different devices in the same scenario, as demonstrated by the user study reported in Section 6.3.

Furthermore, this dissertation has provided a significant contribution to the design of software support for Ubiquitous Interaction that could relieve designers from implementation and programming burdens. It is to be hoped that, by making technical details

transparent, designers could only focus on creative processes and researchers could better understand how people interact — or do not interact — with these designs.

This dissertation has focused on two open issues regarding the development of Ubiquitous Interaction, which are:

- **Issue 1.** Integration of heterogeneous devices.
- **Issue 2.** Lowering the technical expertise for the rapid prototyping of Ubiquitous Interaction.

Advances with respect to issue 1 have been made through the definition of an hardware abstraction layer that makes use of a common data type model. Primitive data types allow to abstract hardware input, and thus acquiring input from a wide range of different devices, independently of the hardware implementation. Moreover, input from different sources can be combined and define, in this way, new input devices from the software composition of existing hardware. Issue 2 has been addressed by providing a comprehensive environment for programming the behavior of heterogeneous devices (issue 1) that are operated by different interaction techniques. A common communication protocol has been used that is based on OSC and it is compliant with TUIO (see description of the protocol on page 111) so that users can easily integrate middlewares for input handling or programming their own, again independently of the underlying hardware components. Encompassing different devices and interaction techniques in a single programming environment, which is the manifestation of a comprehensive reference architecture, has been demonstrated (Chapter 6) to effectively reduce the development time. From one side users do not need to learn different libraries to manage input devices that may expose the same hardware interface. From another side, an integrated environment provides a coherent approach to manage I/O components, which could not be true when using distinct software solutions due to differences on programming languages or implementation choices. For both cases, refer to Chapter 6. The case of programming the behavior of an accelerometer sensor for the Nintendo Wii Remote Controller or the Apple iPad is presented Chapter 6. Moreover, users commentaries are reported after programming tasks using both the proposed framework or specific libraries for the implementation. It has also been shown that the framework can be successfully used to develop specific applications for ubiquitous environments where different I/O technologies are adopted, as in the case of the digitally-augmented product shelf use case (Chapter 6).

In this chapter the contributions of this dissertation are summarized with respect to the two issues introduced in Chapter 3 and potentials for future research are presented. In Section 7.1 the contributions to the HCI research field are described. Finally in Section 7.2 future works that may arise from this dissertation are acquainted.

7.1 Contributions

The main contributions of this work concern to the HCI research field and its intersection with Ubiquitous Computing and Interaction Design. This dissertation contributes to the development of software support for Ubiquitous Interaction with the following main result:

- **A comprehensive framework** based on primitive data types, where the top-level components allow for flexible and generic access to device features. As reported in the previous section, abstracting from heterogeneous devices implementations requires the definition of a high-level data types structure that can describe raw data from hardware devices in a unified and device-independent way. To this end, a data type structure has been defined for communication between devices in an ubiquitous environment. The framework exploits this data model in a layered architecture (Chapter 5) that provides a comprehensive environment to manage interaction with heterogeneous devices and techniques.

In addition to this primary contribution, the dissertation also provided the following results:

- **Requirements.** The definition of requirements is crucial because they connect the software artifact being developed with the problem driving the development. When developing software support for ubiquitous environments, the designer must know the possibilities offered by the environment but also the needs from the point of view of the users and the technology — in this dissertation the technological side has been addressed. The list of requirements, therefore, contributes to the understanding of technological needs and guides the development of new frameworks.
- **A software library for ubiquitous interaction** that employs hardware abstraction to ease the prototyping process. Providing access to devices, sensors and emitters by means of a unified, high-level API results in the support of the rapid prototyping of interactive systems and the reuse of software components in different applications so to reduce their development time and make it possible for developers to quickly explore numerous designs. The software library is the implementation of the model proposed with the framework.
- **A case study** of an interactive system that demonstrates how the software framework can be used to implement actual applications.
- **An experimental evaluation** with users to demonstrate that a comprehensive framework, which exploits hardware abstraction, positively affects the efficiency of prototyping interaction in the case of device ecologies.

7.2 Potential for Future Research

While, in this dissertation, tangible contributions to the development of ubiquitous computing have been provided, there are aspects that have not been considered or fully addressed, as well as directions that can be further explored and might add knowledge to the larger research area. The implementation presented in Chapter 5 has allowed to demonstrate the effectiveness of the framework. Nevertheless it is not exhaustive. At present time, only a restricted set of input devices and sensors are supported, as well as interaction techniques and widgets of the user interface. Even if, with the actual implementation, it was possible to generate a use case involving touch and marker-based interaction, the framework needs to be extended to a wide range of existing devices and techniques. In this way it will be possible to better understand to what extent the framework is suitable for the development of more complex ubiquitous environments. For instance, the framework is going to be exploited in the European project *meSch*, *Material EncounterS with digital cultural Heritage*¹. The project explores do-it-yourself tangible interactives at heritage sites: extending the framework it will be possible to enable interactions between tangible devices and digital interfaces. At the moment a proof-of-concept has been developed that makes use of a transparent multi-touch display: user interface widgets can be displayed on the screen and, at the same time, objects that are behind the screen are visible like, for instance, a piece in a museum. The digital widgets can be coupled with physical output, activating physical behavior of interactive objects, like having the piece rotating by activating a servo motor².

As pointed out throughout the dissertation, the framework focuses on the technical side of the socio-technological system that characterizes ubiquitous interaction. Having an infrastructure that eases the development of device ecologies, where different devices coexist and communicate, allow designers to focus on the human side and understand how to best design such environments in order to support users' needs. A key challenge, therefore, is determining how to exploit the characteristics of individual devices and how these can be combined to best support various human collaborative activities. First examples can be found in the work of Coughlan et al. [2012]. The support to complex gestural interaction is another point that has been left unexplored in this dissertation. Currently the framework only supports basic gestural interaction, such as using a device to point at another device or recognizing touch points on a touch-sensitive surface. Touch or in-air gestures, body movements or motion sensing gestures with portable devices need to be implemented that allow input from heterogeneous sources to be mapped into commands for the user interface independently of the underlying hardware. Approaches such as the specification language proposed by GISpL [Echtler and Butz, 2012] will be

¹The project (2013-2017) receives funding from the European Community's Seventh Framework Programme 'ICT for access to cultural resources' (ICT Call 9: FP7-ICT-2011-9) under the Grant Agreement 600851.

²A sample video of the proof-of-concept can be found at <http://youtu.be/sFIgyOhnSXU>

explored in the attempt to organize unambiguous description of gestural input across a wide range of input devices. As discussed in Chapter 6 this framework demonstrates to reduce the implementation of applications that make use of different input devices, display technologies and interaction techniques. However, further investigation is needed in order to understand how transferable these results are. For example, current APIs are intended to be used by expert programmers. One question is: *"how does the framework can address the needs of researchers or interaction designers with low, or no, programming experience?"*. The development of a visual language, as also suggested by other projects such as *Squidy* [König et al., 2009] or *vvvv*, can be a path to further explore. Finally, the holistic approach promoted by the framework gives designers a different point of view of the design space and the language of the software libraries gives them a novel tool to express their creative power and create new solutions. We, as humans, create tools to accomplish certain tasks, but the tools we create also affect the kind of actions we can perform and our cognitive processes. Therefore, the development of the framework could also be a starting point to further explore to what extent the rationale of a framework influences the type of design solutions that can be produced and, by comparing different frameworks, understand which tool is the best for a particular problem (or a set of problems) and why. The idea that a tool influences the exploration and understanding of a problem space is not new in science. Just to cite one example: between linguistics, cognitive psychologists and philosophers there is a strong debate on the way the language we speak affects our thoughts about the world, which is known as *linguistic relativism* [Swoyer, 2010]. This question has been in large part supported by the empiric evidence that people from different parts of the world and that speak different languages talk about the world differently. Although there are no definitive answers to this subject, some studies demonstrated that, even if it is not the only cause, the native language of a person plays an important role in shaping habitual thought.

References

- Bailey, B. P., Konstan, J. A., and Carlis, J. V. (2001). Demais: designing multimedia applications with interactive storyboards. In *Proceedings of the ninth ACM international conference on Multimedia*, MULTIMEDIA '01, pages 241–250, New York, NY, USA. ACM.
- Baudel, T. and Beaudouin-Lafon, M. (1993). Charade: remote control of objects using free-hand gestures. *Commun. ACM*, 36(7):28–35.
- Baxter, L. (1997). Capacitive sensors design and applications.
- Bazoli, J. (2012). Interfacce utente avanzate per l'internet of things: verso un'integrazione naturale di persone, oggetti e reti sociali. Master's thesis, Università degli Studi di Brescia.
- Bellucci, A., Malizia, A., and Aedo, I. (2011). Tesis: turn every surface into an interactive surface. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '11, pages 1–1, New York, NY, USA. ACM.
- Bellucci, A., Malizia, A., Diaz, P., and Aedo, I. (2010). Don't touch me: multi-user annotations on a map in large display environments. In *Proceedings of the International Conference on Advanced Visual Interfaces*, AVI '10, pages 391–392, New York, NY, USA. ACM.
- Bencina, R. and Kaltенbrunner, M. (2005). The design and evolution of fiducials for the reactivation system. In *3rd International Conference on Generative Systems in the Electronic Arts*.
- Benyon, D., Turner, P., and Turner, S. (2005). *Designing interactive systems: People, activities, contexts, technologies*. Addison-Wesley.
- Bolt, R. A. (1980). "put-that-there": Voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '80, pages 262–270, New York, NY, USA. ACM.
- Bragdon, A., DeLine, R., Hinckley, K., and Morris, M. (2011). Code space: touch+ air gesture hybrid interactions for supporting developer meetings. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 212–221. ACM.
- Brandt, E. and Messeter, J. (2004). Facilitating collaboration through design games. In *Proceedings of the eighth conference on Participatory design: Artful integration: interweaving media, materials and practices - Volume 1*, PDC 04, pages 121–131, New York, NY, USA. ACM.
- Butler, A., Izadi, S., and Hodges, S. (2008). Sidesight: multi-"touch" interaction around small devices. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST '08, pages 201–204, New York, NY, USA. ACM.

- Buxton, B. (2007). *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Buxton, W. (2009). Multi-touch systems that i have known and loved.
- Buxton, W. (2010). A touching story: A personal perspective on the history of touch interfaces past and future. *Symposium Digest of Technical Papers*, 41(1):444–448.
- Buxton, W., Hill, R., and Rowley, P. (1985). Issues and techniques in touch-sensitive tablet input. *SIGGRAPH Comput. Graph.*, 19(3):215–224.
- Carroll, J. M. (2000). *Making use: scenario-based design of human-computer interactions*. MIT press.
- Christensen, M. (2006). *As We May Feel—Interpreting the Culture of Emerging Personal Affective Mobile Media*. PhD thesis, IT University of Copenhagen.
- Clarke, S. (2004). Measuring api usability. *Doctor Dobbs Journal*, 29(5):1–5.
- Cochran, W. G. and Cox, G. M. (1950). Experimental designs. *Soil Science*, 70(2):164.
- Cook, T. D., Campbell, D. T., and Day, A. (1979). *Quasi-experimentation: Design & analysis issues for field settings*. Houghton Mifflin Boston.
- Costanza, E. and Robinson, J. (2003). A region adjacency tree approach to the detection and design of fiducials. In *VVG*, pages 63–69.
- Costanza, E., Shelley, S. B., and Robinson, J. (2003). D-touch: A consumer-grade tangible interface module and musical applications. In *Proceedings of Conference on Human-Computer Interaction (HCI03)*.
- Coughlan, T., Collins, T. D., Adams, A., Rogers, Y., Haya, P. A., and MartíN, E. (2012). The conceptual framing, design and evaluation of device ecologies for collaborative activities. *Int. J. Hum.-Comput. Stud.*, 70(10):765–779.
- Davis, F. D. (1985). *A technology acceptance model for empirically testing new end-user information systems: Theory and results*. PhD thesis, Massachusetts Institute of Technology.
- de la Barré, R., Chojacki, P., Leiner, U., Mühlbach, L., and Ruschin, D. (2009). Touchless interaction—novel chances and challenges. In Jacko, J., editor, *Human-Computer Interaction. Novel Interaction Methods and Techniques*, volume 5611 of *Lecture Notes in Computer Science*, pages 161–169. Springer Berlin / Heidelberg.
- Dey, A. K., Abowd, G. D., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2-4):97–166.
- Dietz, P. and Leigh, D. (2001). Diamondtouch: a multi-user touch technology. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, UIST '01, pages 219–226, New York, NY, USA. ACM.

- Dippon, A. and Klinker, G. (2011). Kinecttouch: accuracy test for a very low-cost 2.5d multitouch tracking system. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces, ITS '11*, pages 49–52, New York, NY, USA. ACM.
- Dourish, P. and Bell, G. (2011). *Divining a digital future: mess and mythology in ubiquitous computing*. MIT Press.
- Dumas, B., Lalanne, D., Guinard, D., Koenig, R., and Ingold, R. (2008). Strengths and weaknesses of software architectures for the rapid creation of tangible and multimodal interfaces. In *Proceedings of the 2nd international conference on Tangible and embedded interaction, TEI '08*, pages 47–54, New York, NY, USA. ACM.
- Echtler, F. and Butz, A. (2012). Gispl: gestures made easy. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*, pages 233–240. ACM.
- Echtler, F. and Klinker, G. (2008). A multitouch software architecture. In *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges, NordiCHI '08*, pages 463–466, New York, NY, USA. ACM.
- Ellis, T. J. and Levy, Y. (2010). A guide for novice researchers: Design and development research methods. In *Informing Science and IT Education Conference*.
- Endres, C., Butz, A., and MacWilliams, A. (2005). A survey of software infrastructures and frameworks for ubiquitous computing. *Mob. Inf. Syst.*, 1(1):41–80.
- Esenther, A., Forlines, C., Ryall, K., and Shipman, S. (2002). Diamondtouch sdk: Support for multi-user, multi-touch applications. *Mitsubishi Electronics Research Laboratory, Report No. TF2002-48*.
- Figuerola, P., Bischof, W. F., Boulanger, P., and Hoover, H. J. (2005). Efficient comparison of platform alternatives in interactive virtual reality applications. *Int. J. Hum.-Comput. Stud.*, 62(1):73–103.
- Fitzmaurice, G. W., Ishii, H., and Buxton, W. A. S. (1995). Bricks: laying the foundations for graspable user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '95*, pages 442–449, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.
- Freeman, D., Hilliges, O., Sellen, A., O'Hara, K., Izadi, S., and Wood, K. (2012). The role of physical controllers in motion video gaming. In *Proceedings of the Designing Interactive Systems Conference, DIS '12*, pages 701–710, New York, NY, USA. ACM.
- Freeman, J. (1975). The modelling of spatial relations. *Computer graphics and image processing*, 4(2):156–171.
- Fukumoto, M., Mase, K., and Suenaga, Y. (1992). "finger-pointer": a glove free interface. In *Posters and short talks of the 1992 SIGCHI conference on Human factors in computing systems, CHI '92*, pages 62–62, New York, NY, USA. ACM.

- Gill, S., Loudon, G., and Walker, D. (2008a). Designing a design tool: working with industry to create an information appliance design methodology. *Journal of Design Research*, 7(2).
- Gill, S., Walker, D., Loudon, G., Dix, A., Woolley, A., Ramduny-Ellis, D., and Hare, J. (2008b). Rapid development of tangible interactive appliances: Achieving the fidelity / time balance. *International Journal of Art and Technology*, 1(3-4):309–331.
- Green, T. R. G., Petre, M., et al. (1996). Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of visual languages and computing*, 7(2):131–174.
- Greenbaum, J. M. and Kyng, M., editors (1991). *Design at Work: Cooperative Design of Computer Systems*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.
- Greenberg, S. (2007). Toolkits and interface creativity. *Multimedia Tools and Applications*, 32:139–159. 10.1007/s11042-006-0062-y.
- Greenberg, S. and Fitchett, C. (2001). Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, UIST '01, pages 209–218, New York, NY, USA. ACM.
- Hall, E. T. and Hall, E. T. (1969). *The hidden dimension*. Anchor Books New York.
- Han, J. Y. (2005). Low-cost multi-touch sensing through frustrated total internal reflection. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, pages 115–118, New York, NY, USA. ACM.
- Harris, A., Rick, J., Bonnett, V., Yuill, N., Fleck, R., Marshall, P., and Rogers, Y. (2009). Around the table: are multiple-touch surfaces better than single-touch for children's collaborative interactions? In *Proceedings of the 9th international conference on Computer supported collaborative learning-Volume 1*, pages 335–344. International Society of the Learning Sciences.
- Harrison, C. (2010). Appropriated interaction surfaces. *Computer*, 43(6):86–89.
- Harrison, C. and Hudson, S. E. (2008). Scratch input: creating large, inexpensive, unpowered and mobile finger input surfaces. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST '08, pages 205–208, New York, NY, USA. ACM.
- Hartmann, B., Klemmer, S. R., Bernstein, M., and Mehta, N. (2005). d.tools: Visually prototyping physical uis through statecharts. In *Extended Abstracts of UIST 2005*. ACM Press.
- Hasan, H. (2003). Information systems development as a research method. *Australasian J. of Inf. Systems*, 11(1).
- Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105.
- Hornecker, E. (2009). Tangible interaction. http://www.interaction-design.org/encyclopedia/tangible_interaction.html.

- Hornecker, E. (2012). Beyond affordance: tangibles' hybrid nature. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*, TEI '12, pages 175–182, New York, NY, USA. ACM.
- Houde, S. and Hill, C. (1997). What Do Prototypes Prototype? *Handbook of Human-Computer Interaction*.
- Igoe, T. and O'Sullivan, D. (2004). *Physical Computing: Sensing and Controlling the Physical World with Computers*. Course Technology PTR, 1 edition.
- Ishii, H. (2008). Tangible bits: beyond pixels. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, TEI '08, pages xv–xxv, New York, NY, USA. ACM.
- Ishii, H. and Ullmer, B. (1997). Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proc. Conf. on Human Factors in Computing Systems (CHI)*, pages 234–241, Atlanta, GA. ACM Press.
- Israel, J. H., Belaifa, O., Gispén, A., and Stark, R. (2011). An object-centric interaction framework for tangible interfaces in virtual environments. In *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction*, TEI '11, pages 325–332, New York, NY, USA. ACM.
- Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., et al. (2011). Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM.
- Johnson, R. E. (1997). Frameworks = (components + patterns). *Commun. ACM*, 40(10):39–42.
- Jonathan Lazar, Jinjuan Heidi Feng, H. H. (2010). *Research Methods in Human-Computer Interaction*. Wiley.
- Jones, W., Spool, J., Grudin, J., Bellotti, V., and Czerwinski, M. (2007). "get real!": what's wrong with hci prototyping and how can we fix it? In *CHI '07 extended abstracts on Human factors in computing systems*, CHI EA '07, pages 1913–1916, New York, NY, USA. ACM.
- Jordà, S., Geiger, G., Alonso, M., and Kaltenbrunner, M. (2007). The reactable: exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, TEI '07, pages 139–146, New York, NY, USA. ACM.
- Jordà, S., Hunter, S. E., Pla i Conesa, P., Gallardo, D., Leithinger, D., Kaufman, H., Julià, C. F., and Kaltenbrunner, M. (2010). Development strategies for tangible interaction on horizontal surfaces. In *Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction*, TEI '10, pages 369–372, New York, NY, USA. ACM.

- Jorda, S., Kaltenbrunner, M., Geiger, G., and Bencina, R. (2005). The reactable*. In *Proceedings of the international computer music conference (ICMC 2005)*, Barcelona, Spain, pages 579–582.
- Jørgensen, A. H. and Myers, B. A. (2008). User interface history. In *CHI '08 extended abstracts on Human factors in computing systems*, CHI EA '08, pages 2415–2418, New York, NY, USA. ACM.
- Jung, H., Stolterman, E., Ryan, W., Thompson, T., and Siegel, M. (2008). Toward a framework for ecologies of artifacts: how are digital artifacts interconnected within a personal life? In *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*, NordiCHI '08, pages 201–210, New York, NY, USA. ACM.
- Kahn, J. M., Katz, R. H., and Pister, K. S. J. (1999). Next century challenges: mobile networking for “smart dust”. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 271–278, New York, NY, USA. ACM.
- Kaltenbrunner, M. and Bencina, R. (2007). reactivation: a computer-vision framework for table-based tangible interaction. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, TEI '07, pages 69–74, New York, NY, USA. ACM.
- Kaltenbrunner, M., Bovermann, T., Bencina, R., and Costanza, E. (2005). Tuio: A protocol for table-top tangible user interfaces. In *Proceedings of the The 6th Int'l Workshop on Gesture in Human-Computer Interaction and Simulation*, Vannes, France.
- Kammer, D., Freitag, G., Keck, M., and Wacker, M. (2010). Taxonomy and overview of multi-touch frameworks: Architecture, scope and features. In *Workshop on Engineering Patterns for Multitouch Interfaces*, Berlin.
- Klemmer, S. (2012). The power of prototyping. <https://class.coursera.org/hci/lecture/preview>.
- Klemmer, S., Newman, M. W., and Sapien, R. (2000). The designer's outpost: a task-centered tangible interface for web site information design. In *CHI '00 extended abstracts on Human factors in computing systems*, CHI EA '00, pages 333–334, New York, NY, USA. ACM.
- Klemmer, S. R. and Landay, J. A. (2009). Toolkit Support for Integrating Physical and Digital Interactions. *Human-Computer Interaction*, 24(3):315–366.
- Klemmer, S. R., Li, J., Lin, J., and Landay, J. A. (2004). Papier-mache: toolkit support for tangible input. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04, pages 399–406, New York, NY, USA. ACM.
- Klokmoose, C. N. (2009). *On Human-Computer Interaction in Complex Artefact Ecologies*. PhD thesis, Department of Computer Science, Aarhus University, Denmark.
- König, W. A., Rädle, R., and Reiterer, H. (2009). Squidy: a zoomable design environment for natural user interfaces. In *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, CHI EA '09, pages 4561–4566, New York, NY, USA. ACM.

- Krafzig, D., Banke, K., and Slama, D. (2005). *Enterprise SOA: service-oriented architecture best practices*. Prentice Hall PTR.
- Kray, C., Nesbitt, D., Dawson, J., and Rohs, M. (2010). User-defined gestures for connecting mobile phones, public displays, and tabletops. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, pages 239–248. ACM.
- Krueger, M. W., Gionfriddo, T., and Hinrichsen, K. (1985). Videoplac—an artificial reality. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '85, pages 35–40, New York, NY, USA. ACM.
- Kunz, A. and Fjeld, M. (2010). *Tabletops - Horizontal Interactive Displays*, chapter From Table–System to Tabletop: Integrating Technology into Interactive Surfaces, pages 51–69. Springer London.
- Kvale, S. (2007). *Doing Interviews*. SAGE Publications.
- Landay, J. A. (1996). Silk: sketching interfaces like crazy. In *Conference companion on Human factors in computing systems: common ground*, CHI '96, pages 398–399, New York, NY, USA. ACM.
- Laufs, U., Ruff, C., and Zibuschka, J. (2010). Mt4j - a cross-platform multi-touch development framework. *CoRR*, abs/1012.0467.
- Lee, J. and Ishii, H. (2010). Beyond: collapsible tools and gestures for computational design. In *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, pages 3931–3936. ACM.
- Lee, J. C. (2008). Hacking the nintendo wii remote. *IEEE Pervasive Computing*, 7:39–45.
- Lee, J. C., Avrahami, D., Hudson, S. E., Forlizzi, J., Dietz, P. H., and Leigh, D. (2004). The calder toolkit: wired and wireless components for rapidly prototyping interactive devices. In *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques*, DIS '04, pages 167–175, New York, NY, USA. ACM.
- Lifton, J. and Paradiso, J. A. (2010). Dual reality: Merging the real and virtual. In *Facets of Virtual Environments*, pages 12–28. Springer.
- Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*.
- Lim, Y.-K., Stolterman, E., and Tenenber, J. (2008). The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas. *ACM Trans. Comput.-Hum. Interact.*, 15(2):7:1–7:27.
- Lin, H. X., Choong, Y.-Y., and Salvendy, G. (1997). A proposed index of usability: a method for comparing the relative usability of different software systems. *Behaviour & information technology*, 16(4-5):267–277.

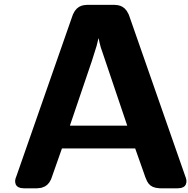
- Lin, J., Newman, M. W., Hong, J. I., and Landay, J. A. (2000). Denim: finding a tighter fit between tools and practice for web site design. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '00, pages 510–517, New York, NY, USA. ACM.
- Loke, S. and Ling, S. (2004). Analyzing observable behaviours of device ecology workflows. In *Proceedings of the 6th International Conference on Enterprise Information Systems*. Citeseer.
- MacIntyre, B., Gandy, M., Dow, S., and Bolter, J. D. (2004). Dart: a toolkit for rapid design exploration of augmented reality experiences. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, UIST '04, pages 197–206, New York, NY, USA. ACM.
- MacKenzie, I. S. (2002). Within-subjects vs. between-subjects designs: Which to use?
- Marquardt, N., Diaz-Marino, R., Boring, S., and Greenberg, S. (2011). The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 315–326. ACM.
- Mazalek, A., Mironer, B., O'Rear, E., and Devender, D. V. (2008). The tvIEWS table role-playing game. *Journal of Virtual Reality and Broadcasting*, 5(8). urn:nbn:de:0009-6-14913,, ISSN 1860-2037.
- Mazalek, A., Reynolds, M., and Davenport, G. (2006). TvIEWS: An extensible architecture for multiuser digital media tables. *IEEE Comput. Graph. Appl.*, 26(5):47–55.
- McLoughlin, D. (1985). A framework for integrated emergency management. *Public Administration Review*, pages 165–172.
- Mellis, D. A., Banzi, M., Cuartielles, D., and Igoe, T. (2007). Arduino: An open electronics prototyping platform. In ACM, editor, *CHI '07 Extended Abstracts*.
- Mi, H. and Sugimoto, M. (2011). Hats: interact using height-adjustable tangibles in tabletop interfaces. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 71–74. ACM.
- Mistry, P. and Maes, P. (2009). SixthSense: a wearable gestural interface. In *ACM SIGGRAPH ASIA 2009 Sketches*, SIGGRAPH ASIA '09, pages 11:1–11:1, New York, NY, USA. ACM.
- Montgomery, D. C. (2008). *Design and analysis of experiments*. Wiley.
- Moscovich, T. and Hughes, J. F. (2006). Multi-finger cursor techniques. In *Proceedings of Graphics Interface 2006*, pages 1–7. Canadian Information Processing Society.
- Muller, M. J. (2001). Layered participatory analysis: new developments in the card technique. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '01, pages 90–97, New York, NY, USA. ACM.
- Muller, M. J. and Kuhn, S. (1993). Participatory design. *Commun. ACM*, 36(6):24–28.

- Mumford, E. (2000). A socio-technical approach to systems design. *Requirements Engineering*, 5(2):125–133.
- Myers, B., Hudson, S. E., and Pausch, R. (2000). Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3–28.
- Myers, B. A. and Rosson, M. B. (1992). Survey on user interface programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '92, pages 195–202, New York, NY, USA. ACM.
- Nam, T.-J. and Lee, W. (2003). Integrating hardware and software: augmented reality based prototyping method for digital products. In *CHI '03 extended abstracts on Human factors in computing systems*, CHI EA '03, pages 956–957, New York, NY, USA. ACM.
- Ni, T. (2011). *A Framework of Freehand Gesture Interaction: Techniques, Guidelines, and Applications*. PhD thesis, Virginia Polytechnic Institute and State University.
- Nielsen, J. (1995). Card sorting to discover the users' model of the information space.
- Nielsen, J. (2005). Time budgets for usability sessions. *Useit. com: Jakob Nielsen's web site*, 12.
- Nielsen, J. and Hackos, J. T. (1993). *Usability engineering*, volume 125184069. Academic press San Diego.
- Noether, G. E. (1987). Sample size determination for some common nonparametric tests. *Journal of the American Statistical Association*, 82(398):645–647.
- Norman, D. A. (1999). *The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution*. The MIT Press.
- Norman, D. A. (2010). Natural user interfaces are not natural. *interactions*, 17(3):6–10.
- NUI (2009). Multi-touch technologies. http://nuigroup.com/log/nuigroup_book_1/.
- O'Hara, K., Harper, R., Mentis, H., Sellen, A., and Taylor, A. (2012). On the naturalness of touchless interaction. *ACM Trans. on Computer Human Interaction*.
- Oulasvirta, A. (2008). Feature: When users "do" the ubicomp. *interactions*, 15(2):6–9.
- Oviatt, S. (2003). Multimodal Interfaces. In Jacko, J. A. and Sears, A., editors, *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, chapter Multimodal interfaces, pages 286–304. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.
- Patten, J., Recht, B., and Ishii, H. (2002). Audiopad: a tag-based interface for musical performance. In *Proceedings of the 2002 conference on New interfaces for musical expression*, NIME '02, pages 1–6, Singapore, Singapore. National University of Singapore.
- Paul, C. L. (2008). A modified delphi approach to a new card sorting methodology. *Journal of Usability Studies*, 4(1):7–30.

- Peffers, K., Tuunanen, T., Rothenberger, M., and Chatterjee, S. (2007). A design science research methodology for information systems research. *J. Manage. Inf. Syst.*, 24(3):45–77.
- Peralta, D. (2012). Master's thesis, Universidad Carlos III de Madrid.
- Perlis, A. J. (1982). Epigrams on programming. *SIgPLAN Notices*, 17(9):7–13.
- Philip, G. C. (1998). Software design guidelines for event-driven programming. *Journal of Systems and Software*, 41(2):79–91.
- Poslad, S. (2009). *Ubiquitous Computing: Smart Devices, Environments and Interactions*. Wiley.
- Potter, R. L., Weldon, L. J., and Shneiderman, B. (1988). Improving the accuracy of touch screens: an experimental evaluation of three strategies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 27–32. ACM.
- Poulton, E. and Freeman, P. (1966). Unwanted asymmetrical transfer effects with balanced experimental designs. *Psychological Bulletin*, 66(1):1.
- Pressman, R. S. (2001). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education, 5th edition.
- Reichl, P., Froehlich, P., Baillie, L., Schatz, R., and Dantcheva, A. (2007). The liliput prototype: a wearable lab environment for user tests of mobile telecommunication applications. In *CHI '07 extended abstracts on Human factors in computing systems*, CHI EA '07, pages 1833–1838, New York, NY, USA. ACM.
- Rekimoto, J. and Matsushita, N. (1997). Perceptual surfaces: Towards a human and object sensitive interactive display. In *Workshop on Perceptual User Interfaces (PUI'97)*, pages 30–32.
- Rekimoto, J. and Saitoh, M. (1999). Augmented surfaces: a spatially continuous work space for hybrid computing environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 378–385, New York, NY, USA. ACM.
- Robertson, J. and Robertson, S. (2012). *Mastering the Requirements Process: Getting Requirements Right*. Addison-Wesley Professional.
- Rogers, Y. (2006). Moving on from weiser's vision of calm computing: Engaging ubicomp experiences. *UbiComp 2006: Ubiquitous Computing*, pages 404–421.
- Saffer, D. (2008). *Designing gestural interfaces*. O'Reilly Media.
- Schöning, J., Brandl, P., Daiber, F., Echtler, F., Hilliges, O., Hook, J., Löchtefeld, M., Motamedi, N., Muller, L., Olivier, P., Roth, T., and von Zadow, U. (2008). Multi-Touch Surfaces: A Technical Guide. Technical report, University of Munich.
- Sculley, J. (1987). *Odyssey: Pepsi to Apple : A Journey of Adventure, Ideas, and the Future*. HarperColins.

- Serrano, M., Nigay, L., Lawson, J.-Y. L., Ramsay, A., Murray-Smith, R., and Deneff, S. (2008). The openinterface framework: a tool for multimodal interaction. In *CHI '08 extended abstracts on Human factors in computing systems*, CHI EA '08, pages 3501–3506, New York, NY, USA. ACM.
- Shapiro, S. S. and Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611.
- Shneiderman, B. (2007). Creativity support tools: accelerating discovery and innovation. *Commun. ACM*, 50(12):20–32.
- Snyder, C. (2003). *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces (Interactive Technologies)*. Morgan Kaufmann, 1 edition.
- Sokal, R. R. (1958). A statistical method for evaluating systematic relationships. *Univ Kans Sci Bull*, 38:1409–1438.
- Streitz, N. A., Geissler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C., Reischl, W., Rexroth, P., Seitz, P., and Steinmetz, R. (1999). i-land: an interactive landscape for creativity and innovation. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 120–127, New York, NY, USA. ACM.
- Stylos, J. and Myers, B. A. (2008). The implications of method placement on api learnability. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 105–112. ACM.
- Swoyer, C. (2010). *Language and Bilingual Cognition*, chapter How does language affect thought? Psychology Press.
- Taylor, II, R. M., Hudson, T. C., Seeger, A., Weber, H., Juliano, J., and Helser, A. T. (2001). Vrpn: a device-independent, network-transparent vr peripheral system. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '01, pages 55–61, New York, NY, USA. ACM.
- Tse, E., Shen, C., Greenberg, S., and Forlines, C. (2006). Enabling interaction with single user applications through speech and gestures on a multi-user tabletop. In *Proceedings of the working conference on Advanced visual interfaces*, pages 336–343. ACM.
- Ullmer, B. and Ishii, H. (1997). The metadesk: models and prototypes for tangible user interfaces. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, UIST '97, pages 223–232, New York, NY, USA. ACM.
- Ullmer, B. and Ishii, H. (2000). Emerging frameworks for tangible user interfaces. *IBM systems journal*, 39(3.4):915–931.
- van Dam, A. (1997). Post-wimp user interfaces. *Commun. ACM*, 40(2):63–67.
- van den Akker, J. (2000). *Principles and methods of development research*. Kluwer Academic Publishers.

- Vaughan-Nichols, S. J. (2009). Game-console makers battle over motion-sensitive controllers. *Computer*, 42:13–15.
- Vogt, W. P. and Johnson, R. B. (2011). *Dictionary of statistics & methodology: A nontechnical guide for the social sciences*. SAGE Publications, Incorporated.
- Wallace, V. L. (1976). The semantics of graphic input devices. In *The papers of the ACM symposium on Graphic languages*, pages 61–65, New York, NY, USA. ACM.
- Weiser, M. (1991). The computer for the 21st century. *Scientific American*.
- Welch, G. and Bishop, G. (1995). An introduction to the kalman filter.
- Wigdor, D., Jiang, H., Forlines, C., Borkin, M., and Shen, C. (2009). Wespace: the design development and deployment of a walk-up and share multi-surface visual collaboration system. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 1237–1246. ACM.
- Wigdor, D. and Wixon, D. (2011). *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.
- Wilcoxon, F. and Wilcox, R. A. (1964). *Some rapid approximate statistical procedures*. Lederle Laboratories.
- Wilde, N. P. (1996). Using cognitive dimensions in the classroom as a discussion tool for visual language design. In *Conference companion on Human factors in computing systems: common ground*, pages 187–188. ACM.
- Wilson, A. D. (2010). Using a depth camera as a touch sensor. In *ACM International Conference on Interactive Tabletops and Surfaces, ITS '10*, pages 69–72, New York, NY, USA. ACM.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer.
- Wu, A., Jog, J., Mendenhall, S., and Mazalek, A. (2011). A framework interweaving tangible objects, surfaces and spaces. In *Proceedings of the 14th international conference on Human-computer interaction: interaction techniques and environments - Volume Part II, HCI'11*, pages 148–157, Berlin, Heidelberg. Springer-Verlag.
- Wu, A., Mendenhall, S., Jog, J., Hoag, L. S., and Mazalek, A. (2012). A nested api structure to simplify cross-device communication. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction, TEI '12*, pages 225–232, New York, NY, USA. ACM.
- York, J. and Pendharkar, P. C. (2004). Human- computer interaction issues for mobile computing in a variable work context. *International Journal of Human-Computer Studies*, 60(5-6):771–797.
- Zarraonandia, T., Aedo, I., and Díaz, P. (2012). Envisioning the transformative role of it in lectures. *Interaction Design and Architecture(s) Journal*, pages 7–17.



Published Results

THIS appendix lists, in reverse chronological order, the scientific publications that resulted from this dissertation.

- Bellucci, A., Malizia, A. and Aedo, I. (2013). Light on horizontal interactive surfaces: input space for tabletop computing. Accepted for publication on ACM Computing Surveys on date 26/06/2013.
- Bellucci, A. (2012). Prototyping natural interaction. *BCS Interfaces*, Autumn 2012, 92, 20-21.
- Bellucci, A., Malizia, A. and Aedo, I. (2012). Towards a framework for the rapid prototyping of physical interaction. In *Proceedings of the 4th International Workshop on Physicality, co-located with British HCI 2012*.
- Bellucci, A., Malizia, A. and Aedo, I. (2011) TESIS: Turn Every Surface into an Interactive Surface. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces (ITS 2011)*.
- Bellucci, A., Malizia, A., Diaz, P. and Aedo, I. (2010). Human-Display Interaction Technology. Emerging remote interfaces for pervasive display environments. *IEEE Pervasive Computing*, 9(2), 72-76.
- Bellucci, A., Malizia, A., Diaz, P. and Aedo, I. (2010). Don't Touch Me: multi-user annotations on a map in large display environments. In *Proceedings of International Working Conference on Advanced Visual Interfaces (AVI 2010)*. Vol. 1, pp. 391-392. ACM Press.
- Malizia, A., Onorati, T., Bellucci, A., Diaz, P., and Aedo, I. (2009). Interactive Accessible Notifications for Emergency Notification Systems. In *Proceedings of the 5th International Conference on Universal Access in Human-Computer Interaction. Applications and Services (UAHCI09)*, LNCS 5616. Springer Berlin Heidelberg, Berlin, Heidelberg, 385-394.

B

Requirements Extraction

IN this appendix are reported: *(a)* the list of initial requirements extracted by the state of the art and the personal experience, *(b)* a typical interview protocol depending on the profile of the interviewee, *(c)* the additional requirements extracted from the interviews, *(d)* the categories defined with the card-sorting exercise and, *(e)* the list of final requirements under categories.

B.1 List of Initial Requirements

1. Support Multi-Touch Interaction [Kaltenbrunner and Bencina, 2007; Laufs et al., 2010; Wu et al., 2012]
2. Support Touch-less/Remote Interaction (free hand/body movements) [O'Hara et al., 2012]
3. Support Motion Sensing Gestural Interaction [Bellucci et al., 2010]
4. Support Tangible Interaction [Echtler and Klinker, 2008; Kaltenbrunner and Bencina, 2007; Klemmer et al., 2004]
5. Support Multi-Surface Interaction [Wu et al., 2012]
6. Support Proxemic Interaction [Marquardt et al., 2011]
7. Support Multimodal Interaction [König et al., 2009; Serrano et al., 2008]
8. Transparent Communications [Taylor et al., 2001]
9. Distributed Communications [Taylor et al., 2001]
10. Automatic Resource Discovery [Dey et al., 2001]
11. Common Communication Protocol [Kaltenbrunner et al., 2005]
12. Common Programming Language [Greenberg, 2007]

13. Separation of Concerns [Laufs et al., 2010; Taylor et al., 2001]
14. Minimize Code Housekeeping [Dumas et al., 2008]
15. Hide Low-Level Coding Details [König et al., 2009]
16. Automate Processes and Tasks when Possible [Dey et al., 2001]
17. Easy to learn API [Wu et al., 2012]
18. Support Multi-touch Devices [NUI, 2009]
19. Support Tangible Devices [Klemmer and Landay, 2009]
20. Support Touchless Devices [Bellucci et al., 2010]
21. Support Traditional Devices [König et al., 2009; Serrano et al., 2008]
22. Support Sensors Input/Output [Hartmann et al., 2005; Mellis et al., 2007]
23. Device Identification/Tracking within the Interactive Space [Wu et al., 2012]
24. User Identification/Tracking within the Interactive Space [Marquardt et al., 2011]
25. Real/Virtual Space Matching [Taylor et al., 2001]
26. Data Fusion Mechanisms [Dumas et al., 2008; Israel et al., 2011; Taylor et al., 2001]
27. Data Fission Techniques [Dumas et al., 2008; Israel et al., 2011; Taylor et al., 2001]
28. Familiar Development Platform [Greenberg, 2007]
29. Programming via API [Wu et al., 2012]
30. Programming via Configuration File [Wu et al., 2012]
31. Visual Programming Tools [Hartmann et al., 2005; König et al., 2009; Marquardt et al., 2011; Serrano et al., 2008]
32. Input Abstraction and Interpretation [König et al., 2009; Laufs et al., 2010; Taylor et al., 2001]
33. Manage Heterogeneous Data Content [König et al., 2009; Laufs et al., 2010; Taylor et al., 2001]
34. Device Interoperability [Laufs et al., 2010; Taylor et al., 2001; Wu et al., 2012]
35. Device Abstraction [Laufs et al., 2010; Taylor et al., 2001]
36. Manage Heterogeneous Devices [Israel et al., 2011; Laufs et al., 2010; Taylor et al., 2001]
37. Manage Different Data Streams [Dumas et al., 2008; Taylor et al., 2001]
38. Distributed Application Architecture [Marquardt et al., 2011; Taylor et al., 2001]

39. Data-Oriented Architecture [König et al., 2009; Serrano et al., 2008; Wu et al., 2012]
40. Time Stamps for I/O Messages [Taylor et al., 2001]
41. Storage and Replay of Interactive Sessions [Dey et al., 2001]
42. Spatial Awareness within the Interactive Space [Marquardt et al., 2011]
43. Full-Duplex (two-way) Communications (Physical Object/Virtual Representation) [Israel et al., 2011]
44. Real (e.g., haptic)/Digital(e.g., visual) User Feedback [Bellucci et al., 2010]
45. Event-Driven Architecture [Laufs et al., 2010]
46. Expandible/Extensible Architecture [Laufs et al., 2010]
47. Full and Clear Documentation of Architecture and API [Wu et al., 2012]
48. Common I/O Data Model [König et al., 2009]
49. Support User Scripting [Greenberg, 2007]
50. Cross-Device UI [Bragdon et al., 2011]

B.2 Interviews

Technical background. Typical questions for respondents with a technical, programming background have been:

- Describe the projects related to the design and development of interactive systems you are currently involved or have participated in the past...
- Have you ever designed and/or developed interactive systems for collaborative environments with heterogeneous technologies (e.g. multi-touch tables, video walls, smartphones, tangible input, etc.)?
- What kind of input devices have you programmed? Which kind of interaction technique?
- What kind of programming languages, IDE, SDK you used in your projects?
- How did you manage the communication and data exchange between different devices?
- Do you know the OSC/TUIO protocol?
- Do you know the Arduino platform?
- Have you found yourself interested in doing something that the programming tool you where using did not support? If so, what?

Non technical background. Typical questions for respondents with an academic background and little if no programming knowledge have been:

- What kind of projects regarding ubiquitous technologies have you been involved in?
- Could you please describe what kind of interactive scenarios were you interested to setup?
- How ubiquitous technologies are supposed to support or enhance envisioned interaction?
- What kind of problems did you encounter in your projects? What was the most time-consuming part of the project?

B.3 Additional Requirements

1. Support Cross-Device Interaction
2. Support Augmented-Reality Interactions
3. Support Visual Markers Input (e.g., qrcode)
4. Easy Configuration of Input Devices
5. Storage and Replay of Interactive Sessions
6. Agnosticism of Legacy Middlewares
7. Low Viscosity of the Code
8. Easy to Program UI Behavior
9. Programming via Hard Coding

B.4 Categories

- Input/Output Hardware
- Data
- Architectural Traits
- Developing/Coding
- Interaction Modalities
- Interactive Space
- Application/User Interface

B.5 Final Requirements Under Categories

Input/Output Hardware

1. Heterogeneous Input/Output Hardware
 - Multi-touch Devices
 - Tangible Devices
 - Touchless Devices
 - Traditional Devices
 - Sensors (e.g., cameras, depth sensors, motion sensors)
 - Visual Markers Input for Augmented Reality (e.g., qrcode)

Interaction modalities

2. Heterogeneous Interaction Modalities
 - Multi-Touch Interaction
 - Touch-less/Remote Interaction
 - Free-Hand, Body Movement Gestural Interaction
 - Device (Motion Sensing) Gestural Interaction
 - Proxemic Interaction
 - Tangible Interaction
 - Multimodal Interaction
 - Cross-device Interaction

Interactive Space

3. Spatial Awareness
 - Matching between Real and Virtual Spaces
 - Device Identification/Tracking within the Interactive Space
 - User Identification/Tracking within the Interactive Space
4. Multi-Surface Environment

Architectural Treats

5. Distributed Architecture
 - Transparent to the User (TTTU) Connections and Communications
 - Automatic Devices Discovery
 - Distributed Communications
 - Full-duplex Communications
6. Common Communication Protocol
 - Device Interoperability
7. Expandible/Extensible Architecture
 - Separation of Concerns
 - Common I/O Data Model
 - Data-Oriented
 - Event-Driven Architecture
8. Device Abstraction
 - Input Abstraction and Interpretation
9. Different Data Streams
 - Data Fusion/Fission Mechanisms
 - Time Stamps for I/O Messages
10. Agnosticism of Legacy Middlewares

Developing/Coding

11. Common Programming Language
12. Familiar Development Platform
13. Low Viscosity of the Code
14. Hide Low-Level Coding Details
 - Minimize Housekeeping
 - Automate Processes and Tasks when Possible
15. Programming Alternatives
 - Programming via API

- Concise API
- Easy to learn API
- Full and Clear Documentation of Architecture and API
- Programming via Hard Coding
- Programming via Configuration File
- Visual Programming Tools
- User Scripting

Application/User Interface

16. Real (e.g., haptic)/Digital(e.g., visual) User Feedback
 - Physical Object/Virtual Representation Communications (bidirectional)
17. Cross-Device UI
18. Storage and Replay of Interactive Sessions
19. Easy Configuration of Input Devices
20. UI Widgets
 - Predefined UI Widgets
 - Common Behavior to Input Methods
 - Easy to customize/extended

C

User Evaluation

In this Appendix the material used during the user study is presented: the documentation given to the users with step-by-step instructions of the evaluation, the informed consent form, the pre-test and post-test questionnaires.

C.1 Documentation for the User Tasks

The following pages show the documentation given to the subject during the experimentation.



Researcher: Andrea Bellucci, Ph.D. student, Dept. of Computer Science, abellucc@inf.uc3m.es.

Supervisor: Ignacio Aedo, Full Professor, Dept. of Computer Science, aedo@ia.uc3m.es.

Evaluation of a comprehensive framework for the rapid prototyping of ubiquitous interaction

D1. User study

You have been asked to participate in an experiment conducted by Andrea Bellucci to investigate the impact of different software technologies in the prototyping of ubiquitous interaction. The results of the experiment will be included in Andrea's Ph.D. thesis. The session will not last more than 90 minutes, in which you will be introduced to the context of the experiment and the technologies that will be used to carry out the tasks. You will be asked to carry out two programming tasks and to fill out two anonymous questionnaires, one at the beginning and one at the end, to gather information on your profile and your experience after the programming tasks. Please be aware that the audio and video of the session will be recorded in a digital archive so that it can be used for data analysis. We do not record the interview without permission: you are kindly asked to read and sign the consent (document D2) before proceeding with the experiment.

Introduction

The Ubiquitous Computing (UbiComp) is considered as an extension of the computational capabilities of the physical environment, allowing the computational infrastructure to be present everywhere in the form of small, inexpensive, robust networked processing devices, distributed at all scales throughout everyday life and generally turned to distinctly common-place ends. In the context of interaction in ubiquitous environments, the trend is to extend the number of different input devices the user is capable to interact with; they can be either traditional ones such as mouse and keyboard, or exploit new types of sensors and tactile surfaces, accelerometers, RGB and infrared cameras, etc. In reality, UbiComp is facing a big divide: due to the heterogeneity of its technologies, practical applications and interfaces are pluggable and interchangeable only to a very limited extent. To cope with this problem, tools that support the rapid prototyping of interfaces that use the aforementioned hardware are needed. The goal of these tools is to develop a richer interaction experience, in a fast and simple way, having the potential to provide time on task by lowering prerequisite knowledge and by automating 'low-level' programming skills.

In this experiment you will be asked to program the behavior of a physical/digital environment using (a) different software libraries and (b) a comprehensive software framework.

Pre-test questionnaire

You are kindly asked to fill out a pre-test questionnaire.

Training

You are given a brief tutorial on the usage of the Processing platform. The tutorial consists of the following steps:

1. An introduction the Processing platform: the language, the API, the concept of Sketch, how to import libraries in Processing and how to use the graphical IDE.
2. Practice: you will program a first “Hello World” sketch to get confident with the platform.

The basic concepts of an accelerometer are explained.

Scenario

In the last two years you have been working in the R&D (Research and Development) department of a company that develop solutions for technologically enhanced environments (e.g., smart rooms with tablets and video wall interfaces). Thanks to your programming skills, you have been proposed to join an inter-disciplinary group (designers, artists, computer scientists, engineers, etc.) with the task to develop prototypes to investigate the use of new sensors (e.g., Microsoft Kinect) and ubiquitous devices (e.g., tablets, smart windows, etc.) for physical/digital interfaces. The sensors you are interested to use in your prototypes are:

- An accelerometer, a digital device capable of measure acceleration in two or three axis of the space.
- Touch sensor
- Buttons

In order to learn how data retrieved from these sensors can be used in your projects, you decide to explore actual devices that make use of an accelerometer, touchscreens and buttons: the Nintendo Wii Remote Controller and the Apple iPad. As development platform you chose to use Processing, because its ease of use, flexibility and widespread diffusion within designers and digital artists. A quick search on the internet reveals that there is a Processing library to interface with the Wiimote (**wrj4P5**), another one to get the acceleration and touch data from the iPad (**oscP5**) and then you also find out that a **framework** for Processing has been developed that allows to get data from sensors, independently from the underlying hardware. The framework supports the Wiimote and the iPad and therefore you think that it can be worthwhile to explore the functionalities of the three software libraries. To this end you start to take a look at the API of each one the libraries: wrj4p5, oscP5 and the framework...

...you now have up to 15 minutes to study the API of the provided libraries: the library to program in processing with an iPad, the Wiimote and the framework. You can stop before the 15 minutes if you feel you have gathered enough knowledge to start programming.

Here you have the URL to the API reference for each library:

- wrj4P5 <http://sourceforge.jp/projects/wrj4p5/wiki/Wrj4P5%28en%29>
- oscP5 <http://www.sojamo.de/libraries/oscP5/reference/index.html>
- framework <http://dei.inf.uc3m.es/abellucci/hat/index.html>

Introduction to programming tasks

Use the two devices (Wiimote and iPad) to interact with a virtual environment and see the results of the two tasks (T1 and T2): press the A button on the Wiimote or touch the screen of the iPad to have the color of a 3D box to change from red to green and move the device to have the 3D box rotate.

Programming Tasks: general description

You've got to program a prototype that makes use of an accelerometer, touch screen (iPad) and buttons (Wiimote) to interact with a virtual 3D box, both with a Wiimote and an iPad. In order to ease the development you are given a template and you are asked to fill the missing code where specified.

- **Programming Task (T1).** Your goal is to **change the color (from red to green) of the virtual 3D box** by pressing the A button on the Wiimote or touch the screen on the iPad. You are going to program the behavior of the 3D box according to the input from the A button of the Wiimote and the touch sensor of the iPad in both of these two scenarios:
 1. Using the two libraries: **wrj4P5** for the Wiimote and **oscP5** for the iPad;
 2. Using a **framework** that provides integration of devices.

The task is divided into the following subtasks:

1. Write the code to establish a connection with the device;
2. Write the code to get relevant data;
3. Use the input data to have the cube changing color.

- **Programming Task (T2).** Your goal is to **rotate a virtual 3D box on the X axis** using the data from the embedded accelerometer (Figure 1 and shows rotation axes of the Wiimote and iPad). You are going to program the behavior of the cube according to the input from accelerometer of the Wiimote and iPad in both of these two scenarios:
 1. Using the two libraries: **wrj4P5** for the Wiimote and **oscP5** for the iPad;
 2. Using a **framework** that provides integration of devices.

The task is divided into the following subtasks:

1. Write the code to establish a connection with the device;
2. Write the code to get relevant data;
3. Use the acceleration data to have the cube rotating on the x axis.

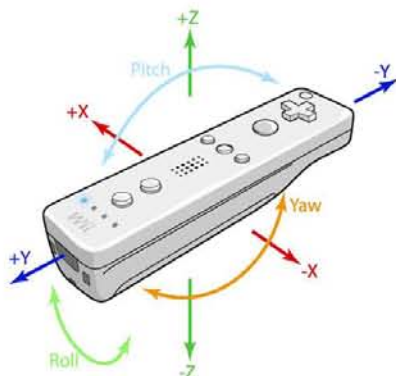


Fig. 1. Wiimote axis.

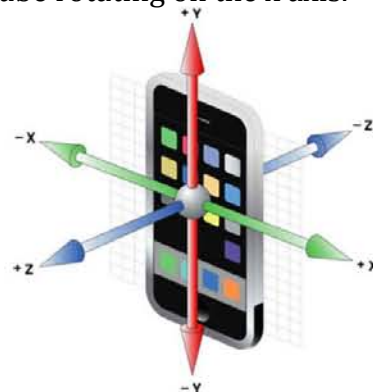


Fig. 2 iPad axis.

Post-test questionnaire

You are kindly asked to fill out a post-test questionnaire.

Debriefing

The experimenter reports on details about the experiment and you are free to ask for further explanations and/or provide any other comment/suggestion you might have.

C.2 Written Consent Form

The following pages show the consent form participants were asked to sign before starting the evaluation session.



Researcher: Andrea Bellucci, Ph.D. student, Dept. of Computer Science, abellucc@inf.uc3m.es.

Supervisor: Ignacio Aedo, Full Professor, Dept. of Computer Science, aedo@ia.uc3m.es.

Evaluation of a comprehensive framework for the rapid prototyping of ubiquitous interaction

D2. Consent to participate in evaluation

You have been asked to participate in a user study conducted by Andrea Bellucci from Computer Science Department at Universidad Carlos III de Madrid. The purpose of the study is to investigate the impact of different software technologies in the prototyping of ubiquitous interaction. The results of this study will be included in Andrea Bellucci's Ph.D. thesis. You should read the information below, and ask questions about anything you do not understand, before deciding whether or not to participate.

- You will not be compensated for this evaluation session.
- Unless you give us permission to use your name, title, and / or quote you in any publications that may result from this research, the information you tell us will be confidential.
- We would like to record the audio and video of this evaluation on a digital archive so that we can use it for reference while proceeding with this study. We will not proceed with any recordings without your permission. If you do grant permission for this session to be recorded, you have the right to revoke recording permission and/or end the session at any time.

Written consent

Your signature on this form imply that (1) you understand the procedures described above, (2) your questions have been answered to your satisfaction and (3) you agree to participate in this study. You will be given a copy of this form.

☐ I give permission for this evaluation session to be recorded on a digital archive.

☐ I give permission for the following information to be included in publications resulting from this study:

- ☐ my name ☐ my title
- ☐ direct quotes from the audio recordings
- ☐ still images of me

Name of Participant _____

Signature of Participant _____ Date _____

Signature of Researcher _____ Date _____

In case of any questions or concerns, please feel free to contact: Andrea Bellucci Ph.D. Student, Dept. of Computer Science, Universidad Carlos III de Madrid Avda. de la Universidad, 30 28911, Leganés, Madrid, +34 91624 6260, abellucc@inf.uc3m.es.

C.3 Pre-test questionnaire

The following pages show the questionnaire participants were asked to fill out before starting the evaluation session. Data from this questionnaire has been used for user profiling.

Pre-test questionnaire

ID: _____

Age: _____

Gender: ☐ Male
☐ Female

Current Workplace (e.g., academia, industry, etc.): _____

Occupation (e.g., Ph.D. student, developer, etc.): _____

1. Of the following programming language and technologies, check those that you have personally used and are familiar with:

- | | |
|------------------------------------|---------------------------------------|
| <input type="checkbox"/> Java2 SE | <input type="checkbox"/> Javascript |
| <input type="checkbox"/> Java2 EE | <input type="checkbox"/> Objective-C |
| <input type="checkbox"/> C/C++ | <input type="checkbox"/> Ruby |
| <input type="checkbox"/> C# | <input type="checkbox"/> Matlab |
| <input type="checkbox"/> Flash AS3 | <input type="checkbox"/> Python |
| <input type="checkbox"/> PHP | <input type="checkbox"/> Visual Basic |

write any other programming language that does not appear in the list: _____

2. Please rate your technical programming knowledge (programming paradigms, data structures, frameworks, etc.)

Low							High
1	2	3	4	5	6		7

3. If you know/use the Java programming language, what is your level of proficiency with the language?

Elementary Proficiency							Professional Proficiency
1	2	3	4	5	6		7

4. How many Operating Systems (OS) have you worked with?

- | | |
|-------------------------------|--------------------------------------|
| <input type="checkbox"/> none | <input type="checkbox"/> 3-4 |
| <input type="checkbox"/> 1 | <input type="checkbox"/> 5-6 |
| <input type="checkbox"/> 2 | <input type="checkbox"/> more than 6 |

5. Of the following programming environment and software frameworks, check those that you have personally used and are familiar with:

- | | | |
|----------------------------------------|------------------------------------------------------------------|------------------------------------------------------------|
| <input type="checkbox"/> Eclipse | <input type="checkbox"/> VVVV | <input type="checkbox"/> Squidy |
| <input type="checkbox"/> Processing | <input type="checkbox"/> Physical Prototyping
(e.g., arduino) | <input type="checkbox"/> Open Exhibits |
| <input type="checkbox"/> Visual Studio | <input type="checkbox"/> Quartz Composer | <input type="checkbox"/> Windows Presentation
Framework |
| <input type="checkbox"/> XCode | <input type="checkbox"/> OpenFrameworks | <input type="checkbox"/> TISCH |
| <input type="checkbox"/> Pure Data | <input type="checkbox"/> Matlab | <input type="checkbox"/> reacTIVision |
| <input type="checkbox"/> Max/MSP | <input type="checkbox"/> Unity3D | |

write any other environment/framework that does not appear in the list: _____

6. If you know/use the Processing development environment, what is your level of proficiency with this environment?

Elementary Proficiency							Professional Proficiency
1	2	3	4	5	6		7

7. Of the following devices and sensors, check those that you have personally used and are familiar with:

- | | | |
|---------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|------------------------------------------------------------------|
| <input type="checkbox"/> desktop/personal
computer | <input type="checkbox"/> joy stick | <input type="checkbox"/> projector |
| <input type="checkbox"/> laptop computer | <input type="checkbox"/> optical stylus | <input type="checkbox"/> mini/pico projector |
| <input type="checkbox"/> tablet device (e.g., iPad) | <input type="checkbox"/> graphics tablet | <input type="checkbox"/> microphone |
| <input type="checkbox"/> smartphone
(e.g., iPhone) | <input type="checkbox"/> head mounted display | <input type="checkbox"/> LED |
| <input type="checkbox"/> touch screen | <input type="checkbox"/> virtual/augmented reality glass | <input type="checkbox"/> Arduino and the like |
| <input type="checkbox"/> interactive surfaces
(e.g., reacTable*) | <input type="checkbox"/> virtual reality glove | <input type="checkbox"/> RGB camera |
| <input type="checkbox"/> keyboard | <input type="checkbox"/> haptic feedback (e.g., rumble) | <input type="checkbox"/> depth camera |
| <input type="checkbox"/> mouse | <input type="checkbox"/> touchless input (e.g., Wiimote) | <input type="checkbox"/> motion sensors
(e.g., accelerometer) |
| <input type="checkbox"/> trackball | <input type="checkbox"/> hand gestures or body
movement recognition (e.g., Microsoft
kinect) | |

write any other device that does not appear in the list: _____

8. Have you ever programmed interactive systems that make use of the aforementioned hardware?

☐ Yes

☐ No

8.1. If you answered yes, would you please specify the device, programming language and its use?

9. Is your academic/professional background related to interaction design, human-computer interaction or ubiquitous computing?

☐ Yes

☐ No

Additional comments: _____

C.4 Post-test questionnaire

The following pages show the questionnaire participants were asked to fill out before starting the evaluation session. Data from this questionnaire has be used to test the framework against the three dimension of *Threshold and Ceiling*, *Predictability* and *Moving Targets* defined by Myers et al. [2000].

Post-test questionnaire

ID _____

General

1. How would you define your experience of programming the interaction of the prototype WITH the framework?

Terrible 1	2	3	4	5	6	Wonderful 7	NA
Frustrating 1	2	3	4	5	6	Satisfying 7	NA
Dull 1	2	3	4	5	6	Stimulating 7	NA
Difficult 1	2	3	4	5	6	Easy 7	NA
Inadequate power to the task 1	2	3	4	5	6	Adequate power to the task 7	NA
Rigid 1	2	3	4	5	6	Flexible 7	NA

2. How would you define your experience of programming the interaction of the prototype WITHOUT the framework (with wrj4P5 and oscP5)?

Terrible 1	2	3	4	5	6	Wonderful 7	NA
Frustrating 1	2	3	4	5	6	Satisfying 7	NA
Dull 1	2	3	4	5	6	Stimulating 7	NA
Difficult 1	2	3	4	5	6	Easy 7	NA
Inadequate power to the task 1	2	3	4	5	6	Adequate power to the task 7	NA
Rigid 1	2	3	4	5	6	Flexible 7	NA

3. When reading code that uses framework's Application Programming Interface (API), was it easy to tell what each section of code does? Why?

4. How did the technical environment (Processing) make the programming tasks?

Difficult							Easy	
1	2	3	4	5	6	7		NA

5. Would you use the framework to develop interactive systems for ubiquitous environments?

Never							Always	
1	2	3	4	5	6	7		NA

Threshold and Ceiling

6. Learning how to use the framework was

Difficult							Easy	
1	2	3	4	5	6	7		NA

7. Your background made the learning

Difficult							Easy	
1	2	3	4	5	6	7		NA

8. Getting started with the framework was

Difficult							Easy	
1	2	3	4	5	6	7		NA

9. The time to learn how to use framework was

Slow							Fast	
1	2	3	4	5	6	7		NA

10. Remembering names and use of constructors, methods and variables for the framework was

Difficult							Easy	
1	2	3	4	5	6	7		NA

11. Did the framework allow you to complete the tasks in a straight-forward manner?

Never							Always	
1	2	3	4	5	6	7		NA

12. The ease of programming with the framework depends on the level of experience

Always							Never	
1	2	3	4	5	6	7		NA

13. Once part of the framework's API is learned, how is to infer the rest of it?

Difficult							Easy	
1	2	3	4	5	6	7		NA

14. The API facilitates the exploration, analysis, and understanding of its components, and the way a developer goes about retrieving what is needed

Disagree							Agree	
1	2	3	4	5	6	7		NA

15. How did the amount of code required for the framework for each subtask in this scenario seem to you?

Too Little							Too Much	
1	2	3	4	5	6	7		NA

16. Please write your comments about how difficult is to learn how to use the framework and how much can be done using the framework here:

Predictability

17. The names used for the framework's objects, methods and variables were

Confusing							Clear	
1	2	3	4	5	6	7		NA

18. Were the names of objects, methods and variables of the framework consistent?

Never							Always	
1	2	3	4	5	6	7		NA

19. Did the APIs of the framework behave as expected?

Never							Always	
1	2	3	4	5	6	7		NA

20. The exploration of features by trial and error was

Discouraging							Encouraging	
1	2	3	4	5	6	7		NA

21. The framework is reliable

Never							Always	
1	2	3	4	5	6	7	NA	

22. Please write your comments about the predictability of the framework here:

Moving Targets

23. With the framework it was possible to create proper interaction for different kind of input devices without changing the underlying infrastructure

Disagree							Agree	
1	2	3	4	5	6	7	NA	

24. Changing the input device (Wiimote and iPad) affected the development WITH the framework

Drastically							Not at all	
1	2	3	4	5	6	7	NA	

25. Changing the input device (Wiimote and iPad) affected the development WITHOUT the framework

Drastically							Not at all	
1	2	3	4	5	6	7	NA	

26. Please write your comments about your experience of developing for different devices in an integrated environment (the framework). To what extent it affected the development?
